Universidad
Carlos III de Madrid

Sistema de provisión en Red para líneas móviles

Autor: Daniel Vicente Cañada

Tutor: Miguel Ángel Patricio Guisado

Dedicado a mi familia, a mis amigos,

A mis compañeros de universidad,

A Miguel Ángel Patricio y a mis compañeros

Del departamento de informática de Accenture

INDICE

TABLA DE FIGURAS

1. INTRODUCCION

1.1. AMBIENTACION

Hace ya más de 35 años que se introdujo un nuevo mecanismo de comunicación telefónica, la telefonía móvil, que en ese momento estaba orientaba exclusivamente a las comunicaciones por voz. Estos primeros dispositivos tenían una cobertura muy reducida y poseía un gran número de carencias lo que provocaba grandes fallos en la comunicación. A partir de los años 80 este sistema se va perfeccionando ya que la calidad y la cobertura van mejorando.

Si nos enfocamos directamente en España los primeros dispositivos móviles aparecieron a partir de la segunda mitad de los años 80, aunque solo estaba enfocado a personas con un gran poder adquisitivo.

Con la llegada de los 90 se produce una gran renovación, cada vez se construyen teléfonos más pequeños y por lo tanto con baterías más pequeñas. A este gran cambio se une la aparición de la primera compañía Telefónica con el objetivo exclusivo de la distribución y facturación de este tipo de dispositivos, Moviline filial de telefónica España. Esta aparición produce un gran impacto en la sociedad lo que provoca que la venta de terminales se vaya disparando exponencialmente con el paso de los años.

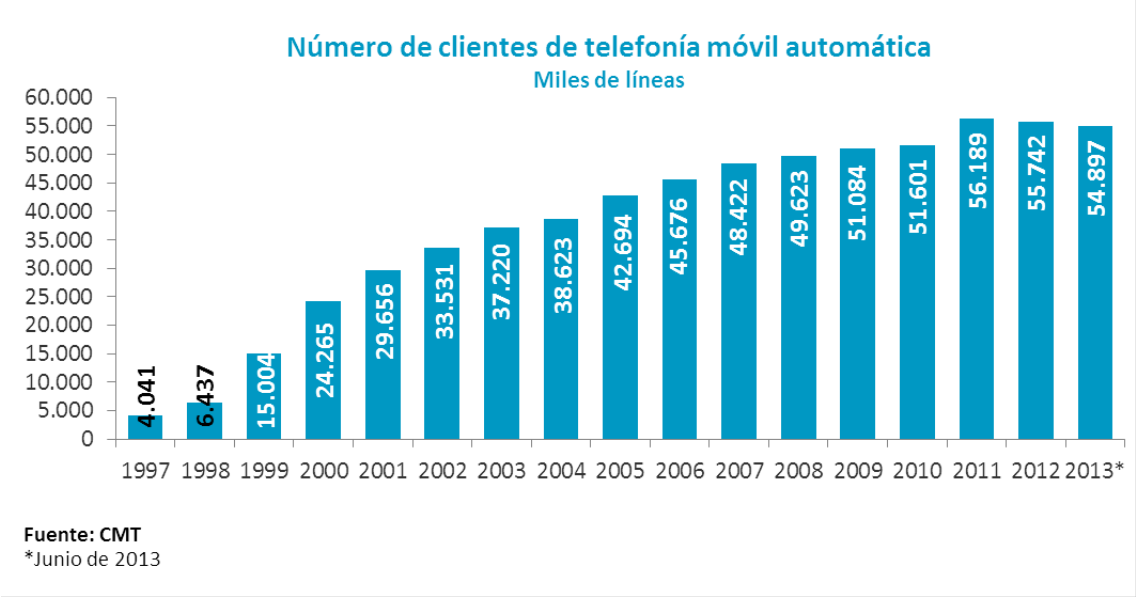


Ilustración 1: Evolución clientes de telefonía

El gran aumento de la venta de líneas móviles provoca que poco a poco vayan apareciendo nuevas compañías telefónicas que nacen con el objetivo de satisfacer las necesidades de un mercado muy determinado. Actualmente se pueden contabilizar más de 30 compañías distintas que ofrecen servicios de telefonía móvil.

Ilustración 2: compañías de telefonía

1.2. MOTIVACION

Analizando los factores relativos al alto crecimiento de líneas móviles existentes con su gran mercado y a la proliferación de nuevas compañías telefónicas veo oportuno la creación de un sistema de provisión de transacciones comerciales enfocadas al terreno de la telefonía móvil.

Debido al gran volumen de operaciones diarias que se realizan en este sector he creído el desarrollo de un sistema que sea capaz de provisionar grandes volúmenes de información en un tiempo limitado, aumentando la seguridad de los datos que maneja y la robustez global del sistema.

Otro de los motivos por lo que he decidido hacer este desarrollo es enfocarlo a nuevas compañías que aparezcan en el mercado ya que se va a desarrollar un sistema de bajo coste lo que favorecerá la implantación del sistema en este tipo de compañías. He tomado esta decisión ya que existen grandes compañías (Movistar, Orange, Vodafone, etc.) que ya tienen grandes sistemas implementados por lo que no creemos que este desarrollo al tratarse de algo nuevo pueda en un primer momento competir con estos “monstruos”.

1.3. OBJETIVOS

El objetivo principal de este proyecto es la realización de un sistema de provisión de datos en red para líneas móviles. El sistema debe ser capaz de gestionar las siguientes tramitaciones:

- Altas y bajas de línea.
- Gestión de servicios (altas, bajas y modificaciones).
- Alta de líneas multisim (líneas con varios aparatos asignados).
- Consulta de líneas.
- Consulta de servicios.

Para la solución del proyecto nos hemos planteado la necesidad de dividir las tramitaciones en dos grandes bloques: por un lado todas aquellas tramitaciones que impliquen una modificación en BBDD (altas, bajas y modificaciones). Para estas tramitaciones el objetivo es crear un sistema que sea capaz de procesar un alto volumen de movimientos manteniendo la seguridad de la información transferida. Para ello nos hemos marcado la realización de un procesamiento Batch (por lotes) que nos permitirá las tramitaciones de alto volúmenes de trabajo.

Por otro lado tenemos la parte relativa a las consultas. Para estos sistema nos hemos marcado la necesidad de crear un sistema de consultas online debido a la necesidad ASAP del usuario en la obtención de los resultados. Vamos a implementar un servidor Tuxedo lo que nos permitirá unos tiempos de respuesta mínimos manteniendo la robustez de los datos.

Para la gestión de datos se propone, tanto para líneas como para tarjetas, el mantenimiento de los mismos llevando un registro de líneas y tarjetas libres para impedir incompatibilidades. Esta gestión será llevada mediante el esquema de base de datos que se decida crear. Así mismo un mismo cliente tendrá la opción poseer varias líneas (multisim) asociadas a distintas tarjetas por lo que será necesario crear un almacenamiento correcto para este objetivo.

Debemos realizar un sistema fiable, no dejar huecos funcionales, manteniendo la integridad de datos. Debido a que nuestra solución no consta de un interfaz directo con el usuario debemos adaptar nuestro sistema a las necesidades del frontal con el que conectaremos creando unos interfaces adaptativos y de fácil uso.

1.4. ESTADO DEL ARTE

En este apartado queremos explicar las tecnologías actuales que están siendo usadas en este sector, resaltando sus defectos y virtudes, defendiendo nuestra elección con respecto a otras posibles soluciones. Así mismo vamos a realizar un pequeño estudio de las distintas tecnologías a usar (Tuxedo, ORACLE, ASN.1, etc...) identificando sus principales características, virtudes, defectos y posibles competidores.

1.4.1.SISTEMAS ACTUALES

En lo relativo al estado del arte, el tener la experiencia del funcionamiento de sistemas similares dentro del mismo sector nos ha hecho inclinarnos por la solución que nosotros consideramos más eficiente. La mayoría de las grandes empresas dentro de este sector (Vodafone, Orange, Movistar, etc...) han desarrollado un sistema de provisión de red encargado de direccionar las peticiones llegadas desde los distintos frontales a BBDD u otras plataformas de red necesarias. Hay casos de compañías (como Tuenti, Pepephone, etc…) que no implementan este tipo de sistemas si no que se valen de sistemas de otras empresas. Consideramos que este planteamiento no es el óptimo debido al gran coste que supone el uso de estos sistemas externos, como también la necesidad de incluir multitud de validaciones internas necesarias antes de enviar peticiones a sistemas externos. Nosotros vemos mejor desarrollar nuestro propio sistema de provisión que tenga la posibilidad de realizar estas validaciones y provisionar la información a BBDD y plataformas sin necesidad de terceros dentro del flujo de operaciones.

Actualmente este tipo de sistemas están implementados en 2 grandes bloques:

- Por un lado nos encontramos con las operaciones que suponen una modificación de datos (bien sea de cara a inserciones, eliminaciones o modificaciones) en BBDD o plataformas.
- Por otro lado nos encontramos con las operaciones de consulta. Debido a que el usuario espera una respuesta inmediata la implementación de estas consultas se debe hacer con sistema de tratamiento ONLINE (procesamiento interactivo).

En lo concerniente al primer punto la gran mayoría de empresas han implementado un sistema por lotes (BATCH) debido al gran volumen de datos que se deben procesar (una de las grandes ventajas de este tipo de procesamiento) y no necesitan de una supervisión directa. Para la parte de consultas la implementación necesaria es una de procesamiento en tiempo real ya que la respuesta debe ser inmediata. Para esta parte nos hemos decidido a implementar una solución Tuxedo bajo C++.

Actualmente la implementación por lotes está muy homogeneizada en la mayoría de empresas por lo que nos basaremos en sistemas ya existentes, intentando mejorar la parte de interfaz intentando implementar un sistema que aporte más robustez y seguridad a los existentes en la actualidad. Creemos que el punto diferencial es la parte de procesamiento online (Tuxedo). Este es un tipo de solución muy poco extendida actualmente pero que consideramos altamente eficiente debido en gran medida a su robustez y su velocidad.

1.4.2.DEFINICION DE LAS TECNOLOGIAS

En este apartado vamos a desarrollar una breve explicación de las distintas tecnologías utilizadas en nuestro sistema, remarcando las principales características, así como una serie de ventajas y desventajas. También hemos querido mostrar una serie de alternativas a dichas tecnologías que han sido desechadas finalmente.

1.4.2.1. LENGUAJE C

El lenguaje C es un lenguaje que se encuentra ubicado en un nivel intermedio entre los lenguajes de alto nivel (PHP, Java,…) y los de bajo nivel (ensamblador) por lo que proporciona una versatilidad entre los lenguajes más cercanos a la forma de pensar humana y los lenguajes de procesamiento a nivel de acceso a memoria.

Se trata de un lenguaje totalmente independiente del hardware del sistema por lo que favorece su movilidad y migración a otros sistemas.

Se ha seleccionado un lenguaje C ya que permite una serie de características que favorecen claramente a nuestro sistema:

- Es un lenguaje flexible lo que propicia varios estilos de programación, aunque el más usado es el estructurado.
- Se trata de un lenguaje de preprocesador por lo que permite la inclusión de múltiples librerías y macros de código fuente que facilitan la ejecución.
- Fácil acceso a la memoria del sistema a través de punteros.
- Posibilidad de interrumpir el procesado en cualquier momento del desarrollo.
- Posibilidad de realizar operaciones en BBDD a través de un lenguaje embebido (Pro *C).

Dentro del mercado sobre el que hemos enmarcado nuestro desarrollo el lenguaje C es más utilizado, por encima Java al que podríamos catalogar como la otra opción válida. Finalmente nos hemos decidido por C principalmente por la sincronización con el lenguaje Pro*C que hemos utilizado en la implementación de los accesos a BBDD. El Pro*C es un lenguaje que utiliza código C o C++ pero que es capaz de "incrustar" código SQL y permite la implementación de cursores. Para la utilización de estas funciones de acceso a BBDD era imprescindible el uso del lenguaje C. Como se indica la otra opción era el C++ pero no lo veíamos adecuado en tema de rendimiento.

1.4.2.2. BBDD ORACLE

Es un sistema de administración de base de datos relacionales, por lo que se realiza el uso de tablas bidimensionales para el almacenamiento de datos.

El sistema permite la integración con varios lenguajes de programación lo que supone una gran ventaja. Para nuestro sistema vamos a utilizar el lenguaje Pro * C, lenguaje que consiste en la realización de operaciones en base de datos (consulta, inserciones, borrados, etc.) integradas en el propio código C, lo que favorece la integración de la base de datos dentro de nuestra aplicación.

Entre las principales características de las bases de datos ORACLE podemos destacar:

- Gran rendimiento y utilización de recursos disponibles.
- Se trata del sistema de base de datos más utilizado en el mundo lo que favorece la expansión de cualquier sistema que este implantado bajo esta tecnología.
- Es el sistema gestor de base de datos más orientado a INTERNET.
- La utilización de un lenguaje de base de datos muy completo lo que permite la creación de diseños completos y muy robustos.
- Es posible desarrollar esta base de datos en diferentes sistemas operativos (flexibilidad).

- Optimización de búsquedas gracias al uso de cursores y de claves primarias y foráneas.
- Correcto soporte.
- Alto rendimiento transaccional.

Como posibles alternativas consideramos la inclusión de MySQL, pero vimos una serie de ventajas que compensaban el gasto económico de implantar un sistema ORACLE:

- Debido al gran mercado existente en este sector, el volumen de datos que deben ser manejados puede ser enorme y crecer exponencialmente, precisamente ORACLE está orientado al manejo de amplios volúmenes de datos. MySQL es una herramienta más orientada a BBDD de un volumen reducido, como podría ser la BBDD de un sitio web.
- Oracle facilita la implementación de una serie de consultas complejas gracias al motor de desarrollo sustentado sobre PL/SQL.
- Desde el punto de vista de seguridad y backup, sin duda alguna el sistema ORACLE proporciona unas soluciones mucho más adecuadas.

1.4.2.3. TUXEDO

Tuxedo es un sistema diseñado para Unix para la realización de un sistema de transacciones CLIENTE – SERVIDOR.

1.4.2.3.1. CONCEPTO DE SISTEMA TRANSACCIONAL

Podemos definir un sistema transaccional como aquel sistema que es capaz de procesar todos los eventos recibidos de forma interactiva (con datos en tiempo real). Los datos que se manejan en las transacciones deben ser consistentes, deben mantener un equilibrio entre los datos solicitados y los existentes en BBDD. Además otra de las ventajas que aporta es que permite la marcha atrás de una tramitación en el momento que exista un fallo, evitando las inconsistencias de datos. Los sistemas transaccionales tienen una serie de ventajas que creemos vitales en nuestro desarrollo:

- La robustez que aporta al sistema: facilidad de marcha atrás, consistencia de datos, son aspectos vitales a la hora del manejo de alto volumen de datos.
- Rapidez: Esta quizá sea la gran ventaja que aportan este tipo de sistemas de cara a nuestro desarrollo. Necesitamos que nuestro sistema sea capaz de tramitar el máximo número de peticiones de forma inmediata.

1.4.2.3.2. TIPOS DE TRANSACCIONES IMPLEMENTADAS

En tuxedo existen 4 grandes tipos de transacciones:

- Comunicación Síncrona: cuando un cliente realiza una petición se queda esperando respuesta antes de continuar con la ejecución.
- Comunicación Asíncrona: el cliente no espera la respuesta y puede continuar con el resto de operaciones, sin embargo tiene la cualidad de poder recuperar dicha respuesta en el momento que lo desee el cliente.
- Comunicación Conversacional: es el caso en que una comunicación Asíncrona realiza más de una tramitación (petición/respuesta) a la vez solo en una dirección.

- Comunicación mediante colas: se utiliza para transacciones que no dependen del momento en que se ejecutan. Es muy usado para procesos batch.

1.4.2.3.3. TIPOS DE DATOS

Existen 4 tipos de datos en Tuxedo:

- Básicos: los datos simples ya conocidos (enteros, cadenas, caracteres, etc.).
- Array de distintos tipos: p.e. array de cadenas.
- FML: archivos de buffers creado para grandes volúmenes de datos.
- XML: documentos .xml enfocados para frontales web.

1.4.2.3.4. PASOS PARA CREAR UN SERVICIO DENTRO DE UN SERVIDOR

➢ De cara al servidor

- Crear el IDL
- Compilarlo para la creación del archivo de código.
- Modificar el archivo de código para incorporar la nueva funcionalidad.
- Recompilar el archivo para la generación del binario

➢ De cara al cliente

- Definir las variables de Tuxedo.
- Conectarse con el servidor externo al que se van a realizar las peticiones.
- Invocación del nuevo servicio.
- Gestión de la respuesta de las tramitaciones.
- Liberación de recursos.

1.4.2.3.5. PASOS PARA LA ADMINISTRACION DE UN SISTEMA TUXEDO

- Definir las distintas variables de entorno que sean necesarias.
- Modificar el archivo UBBCONFIG de cara a añadir la configuración necesaria para el nuevo servidor (identificación de la máquina a la que pertenece el servidor (hostname), definir el nuevo servidor, directorios de la máquina (logs, binarios, etc.)).

- Recompilar el archivo UBBCONFIG con los nuevos cambios.
- Levantar la aplicación (tmboot -y).

1.4.2.3.6. VENTAJAS TUXEDO

- Implementación del modelo cliente- servidor en tres capas.
- Una única conexión con el SGBD por cada servidor y no por cada cliente.
- Permite aplicaciones distribuidas y escalables.
- Permite distintas formas de acceso a los servicios y métodos de comunicación más complejos.
- Es compatible con varias plataformas y sistemas operativos.

1.4.2.3.7. DESVENTAJAS TUXEDO

- Licencia necesaria.
- Se necesita personal especializado para introducir esta tecnología.
- solo implementados para C, C++ y COBOL.

1.4.2.3.8. TECNOLOGIAS COMPETIDORAS

Existen una serie de tecnologías, aparte de Tuxedo, que compiten en este mercado. Se pueden destacar unos ejemplos:

- CICS (IBM): Esta es la gran competencia de Tuxedo. Tecnología enfocada a desarrollos en COBOL Y PL/I.
- Top End (NCR -> BEA): Tecnología que empieza a estar obsoleta. La mayoría de clientes de esta tecnología están siendo migrados a Tuxedo por BEA.

1.4.2.4. ASN.1

ASN.1 es una notación desarrollado para la capa de presentación (capa que define el modo en que los datos van a ser almacenados). Ofrece un amplio conjunto de datos y estructuras de tal forma que se facilita la definición de sistemas complejos mediante la unión de tipos y estructuras simples de datos.

1.4.2.4.1. FORMATO DEL MENSAJE

Los mensajes en ASN.1 están formados por tres campos:

- Campo tipo: en este campo se especifica el tipo de dato que tiene una variable declarada.
- Campo longitud: en este campo viene especificada la longitud del valor que tiene una variable.
- Campo valor: en este campo se especifica el valor asociado a una variable.

1.4.2.4.2. TIPOS DE DATOS

En ASN.1 están definidos tres grandes tipos de datos:

- Primitivos: (datos simples a través de los que se pueden ir construyendo estructuras más complejas): INTEGER, BOOLEAN, OCTECT STRING, OBJECT IDENTIFIER, ENUMERATED y NULL. como ocurre con otros lenguajes por convención se ha decidido que los tipos comiencen con una letra mayúscula.
- Constructores: tipos encargados de generar listas y tablas. Algunos ejemplos de este tipo de datos son SET, SEQUENCE, CHOICE, SET OF y SEQUENCE OF.
- Tipos definidos: son nombres consecutivos de elementos ASN.1 tanto simples como complejos. Suelen ser más descriptivos. Algunos ejemplos de este tipo de datos son: NetworkAddress, IpAddress, Counter, Gauge, TimeTicks, y Opaque.

1.4.2.4.3. MACROS

En ASN.1 existen MACROS que no es más que una definición estructurada de una colección de datos:

bajaLinea OBJECT-TYPE → Declaración del tipo

SYNTAX BajaLinea → Nombre del objeto

ACCESS read-write → Tipo de acceso a la información

STATUS mandatory → Estado del objeto

::=2 → Orden

1.4.2.4.4. TABLA DE CARACTERES ESPECIALES ASN.1

Elemento	Nombre
-	Número con signo
--	Comentario
::=	Asignación ("definido como...")
|	Alternativa (opciones de una lista)
{}	Inicio y final de lista
[]	Inicio y final de una etiqueta (tag)
()	Inicio y final de una expresión de subtipo
..	Indica un rango

Ilustración 3: Carácteres especiales ASN.1

1.4.2.4.5. CODIFICACION DE DATOS

De cara a la codificación de la información cabe indicar que cada tipo de datos tiene una codificación propia establecida en el estándar ISO 8825-1. Por este motivo solo vamos a indicar algunos tipo, a modo de ejemplo, y como se realiza su codificación.

- Codificación tipo INTERGER: Este tipo esta codificado en varios octetos. Cada octeto consiste en números binarios complemento a 2 iguales al valor entero. Se pueden usar tantos octetos como sea necesario.
- Codificación tipo OCTET STRING: Igual que sucede con el tipo INTEGER la codificación de este tipo se realiza a través de una serie ordenada de octetos precedidos por una primitiva que indica que se trata de un tipo OCTET STRING.
- Codificación tipo NULL: este tipo asigna un valor nulo a la variable a la que este asociado. Estará formado primero por una cabecera que indica el tipo de dato, a continuación se indicará la longitud que en este caso será 0 y por último el valor que vendrá vacio ya que este tipo indica un campo sin valor.

1.4.2.4.6. VENTAJAS CODIFICACION ASN.1

Entre las principales ventajas de esta codificación se pueden identificar las siguientes:

- El poder crear, a partir de tipos simples de datos, tipo complejos y una serie de macros.
- ASN. 1 es una codificación muy adaptable a la gran mayoría de lenguajes potentes existentes en el mercado.
- Este sistema de codificación está adaptado perfectamente a protocolos de comunicación de red como TCP-IP.
- La implementación de un interfaz en ASN. 1 es bastante simple debido a la gran cantidad de librerías existente que facilitan enormemente su desarrollo.

1.5. CONTENIDO DE LA MEMORIA

Para poder ver claramente el contenido de esta memoria vamos a especificar los distintos puntos a tratar con una breve descripción de lo que van a contener dichos puntos.

- INTRODUCCION

En este paso hacemos un pequeño comentario sobre el estado del mercado al que afecta nuestro proyecto. Se desarrollarán las distintas motivaciones que nos han afectado a la hora de desarrollar este estudio, las carencias del mercado en este tema. También se mostrarán unos objetivos que han sido previamente fijados de cara a su cumplimiento. En este apartado también se incluye un pequeño estudio en lo relativo a las distintas funcionalidades de las que va a constar nuestro proceso, todas las funcionalidades que va a abarcar, los distintos requisitos técnicos y de software necesarios para el desarrollo e implantación de nuestra aplicación. Por último se especificará una breve introducción a los plazos para la realización del proyecto y el coste estimado del mismo.

- ESPECIFICACION DE REQUISITOS

En este apartado se hará una descripción de los distintos requisitos que se esperan conseguir con este desarrollo. Se especificarán los distintos tipos de requisitos del sistema indicando los roles generales y creando el consiguiente diagrama de casos de uso con las distintas funcionalidades del sistema.

- DISEÑO

En este punto nos centraremos en cómo vamos a diseñar nuestro sistema de cara a satisfaces los distintos requisitos planteados anteriormente. Se especificará por un lado la arquitectura del sistema de cara a establecer un modelo de funcionamiento, que lenguajes vamos a utilizar para la implementación de nuestro sistema, que tipo de conectividades vamos a crear, especificación de los distintos interfaces a utilizar. También haremos una breve descripción de las distintas tecnologías que vamos a utilizar de cara a que el usuario entienda en que nos hemos basado para la elección de

estas tecnologías sobre otras opciones. Por otro lado también mostraremos la arquitectura de la BBDD que vamos a crear, indicando las distintas relaciones, las tablas que vamos a crear con sus campos, los supuestos semánticos y la relación de las distintas tablas dentro del sistema.

- IMPLEMENTACION

En este apartado se mostrará las distintas soluciones empleadas para resolver el proyecto. Se indicará a través de código los distintos procesos implementados, los interfaces creados, las funciones implementadas para el acceso a BBDD. También se incluirá una pequeña guía de uso que mostrará como configurar el sistema de cara a la instalación del sistema.

- PRUEBAS

En este punto se especificarán las distintas pruebas que se van a realizar sobre el sistema. Las dividiremos en cuatro grandes grupos: pruebas unitarias, pruebas de conexión y acceso, pruebas integradas y pruebas de regresión. Se plantearán las distintas pruebas indicando el modo de lanzamiento, el proceso que debe seguir la tramitación y se deberán definir las evidencias que se esperan obtener.

- CONCLUSIONES Y FUTURO TRABAJO

En este apartado lo primero que queremos mostrar son los objetivos iniciales que planteamos especificando si se han cumplido los plazos, si todos los hitos marcados se han desarrollado adecuadamente, problemas que hayan surgido a la hora del desarrollo y como han sido solucionados. Por último también se especificarán futuras líneas de desarrollo de cara a una evolución de la aplicación y como se podría solucionar estas evoluciones. Para finalizar este proyecto se incluirá una breve opinión personal relativa a si estoy satisfecho con este diseño, apartados a mejorar, posibles soluciones alternativas, etc.

2. DISEÑO

2.1. DEFINICION DEL SISTEMA

Para la realización de este sistema se ha optado por una tecnología cliente - servidor.

"La arquitectura cliente-servidor es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta. Esta idea también se puede aplicar a programas que se ejecutan sobre una sola computadora, aunque es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras."

Para el proyecto que nos ocupa el cliente será cualquier usuario que realice una operación a través del portal web. Estos usuarios serán tanto las propias personas que realicen las operaciones con su línea, como los dependientes de las distintas tiendas de la empresa y los operadores que se encargan de las peticiones telefónicas.

Al ser un desarrollo que necesita de un servidor de base de datos como una conexión con plataformas de almacenado de datos será necesario una tecnología de tres capas : cliente – servidor – servidor de almacenamiento de datos.

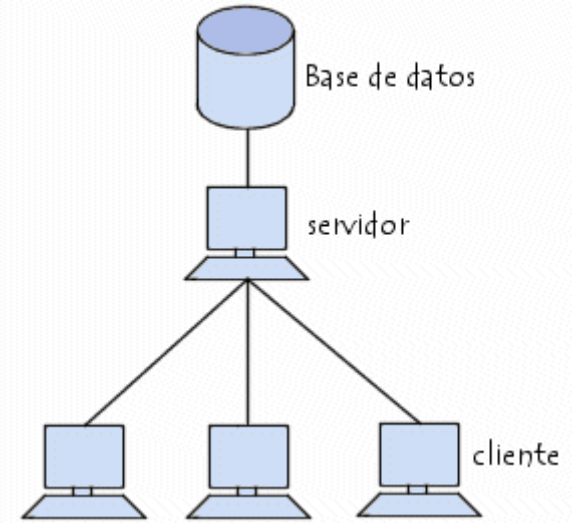


Ilustración 4: arquitectura cliente-servidor

2.1.1. SERVIDOR

Para la elección del servidor se ha optado por un servidor Linux Server, ya que es un servidor que se adapta perfectamente a las características de nuestro sistema.

De cara al interfaz entre el portal web y el gestor se ha optado por dos variantes dependiendo del tipo de operación:

- Para las operaciones de altas y bajas tanto de líneas como de servicios y cambios de teléfono se ha optado por una conexión TCP/IP con una codificación mediante la norma ASN.1:

"ASN.1 es una norma para representar datos independientemente de la máquina que se esté usando y sus formas de representación internas. Es un protocolo de nivel de presentación en el modelo OSI. El protocolo SNMP usa el ASN.1 para representar sus objetos gestionables ".

- Para la parte de consultas al ser una operación en tiempo real se ha optado por procesos online que serán desarrollados mediante la invocación de servicios Tuxedo

"Tuxedo es una plataforma de middleware usada para gestionar procesos transaccionales distribuidos en entornos de computación distribuida. Tuxedo es un middleware orientado a transacciones".

Para la parte de sistema de base de datos se ha optado por una distribución ORACLE, ya que a pesar de tener un coste elevado encaja perfectamente a las características de nuestro sistema debido a la incorporación de servidores Tuxedo.

La parte central de nuestro sistema estará creada mediante procesos Batch:

"Se conoce como sistema por lotes o modo batch, a la ejecución de un programa sin el control o supervisión directa del usuario (que se denomina procesamiento interactivo). Este tipo de programas se caracterizan porque su ejecución no precisa ningún tipo de interacción con el usuario.

Generalmente, este tipo de ejecución se utiliza en tareas repetitivas sobre grandes conjuntos de información, ya que sería tedioso y propenso a errores realizarlo manualmente. Un ejemplo sería el renderizado de los fotogramas de una película".

Estos procesos serán diseñados en lenguaje C. Para la parte de acceso a BBDD de estos procesos se optará por un lenguaje C embebido (Pro* C).

2.2. ARQUITECTURA DEL SISTEMA

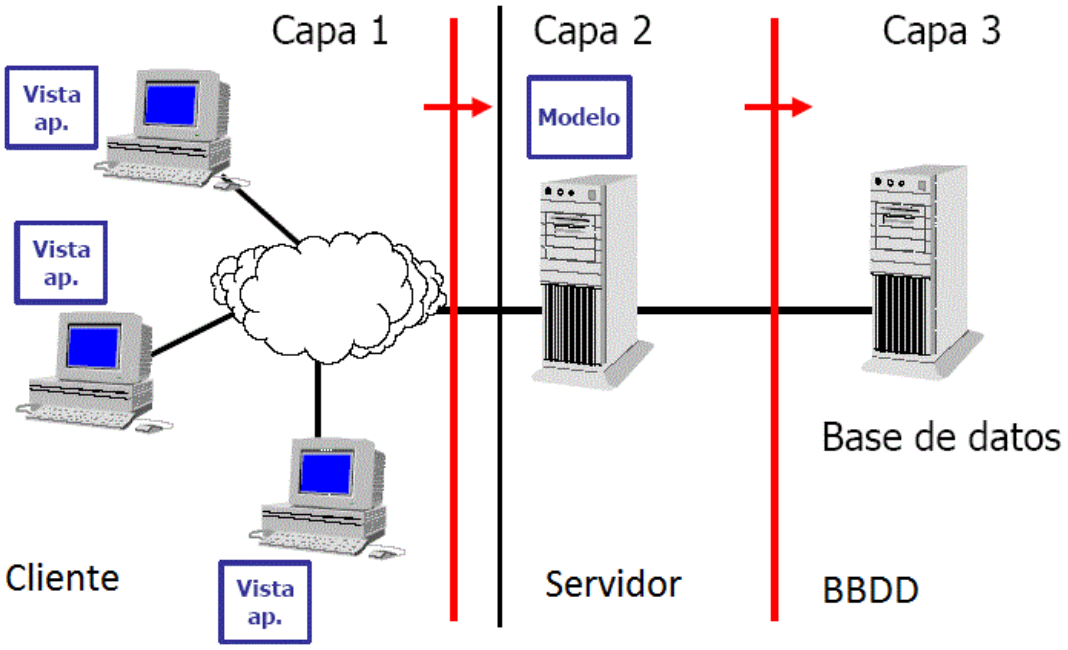


Ilustración 5: arquitectura final del sistema

Como vemos en la ilustración de arriba nuestro sistema estará dividido en 3 capas:

- En una primera capa residirá la parte cliente. Esta parte estará compuesta por un frontal web mediante el cual el cliente llevará a cabo las tramitaciones deseadas. Esta parte conectará directamente con la capa 2 que explicaremos más adelante. Hay que indicar que en este desarrollo no incluimos una parte cliente y nos centraremos en el desarrollo de las otras dos capas, por lo que se debe partir de un frontal ya desarrollado.
- En la segunda capa se encontrará ubicado toda la lógica de nuestro sistema, es la parte del servidor. Esta capa será la encargada de recibir una tramitación, evaluarla y ejecutarla según corresponda. Como hemos comentado más arriba esta capa está implementada en dos tecnologías distintas, por un lado la parte que ejecuta tramitaciones estará implementada en lenguaje C, con código Pro*C para los accesos de base de datos. Por otro lado para la parte de consultas se ha implementado un servidor Tuxedo desarrollado en C++. Los

interfaces entre la capa 1 y la capa 2 (cliente y servidor) están implementados mediante el protocolo ASN.1.

- Por último tenemos la capa 3. Esta capa está formada por la base de datos. Esta BBDD está implementada bajo un sistema ORACLE. Nos hemos inclinado por usar este sistema debido al deseo de implementar los Pro*C para la gestión de los accesos a BBDD. Hemos realizado esta solución ya que pensamos que el flujo de clientes puede ser elevado y los tiempos de reacción en los accesos son mucho más reducidos.

2.3. REQUISITOS FUNCIONALES

Para la especificación de los requisitos del sistema se ha optado por el desarrollo mediante tres grupos. En primer lugar indicaremos los requisitos funcionales del sistema. Esta es la parte en la que se describe la funcionalidad que va a tener el sistema, todas las operaciones que se van a implementar, la manera en que debe ser implementada, los condicionantes, etc. El segundo bloque está compuesto por los requisitos de interfaz. Estos requisitos describen el modo en que se van a conectar los distintos sistemas (protocolos de conexión, codificaciones, etc.). Por último en la última sección se indicarán los requisitos del sistema donde se van a especificar los distintos requisitos de capacidad del sistema (sistema operativo, tecnologías, sistema de base de datos, etc.).

2.3.1. REQUISITOS FUNCIONALES

- RF001 – ALTA NUEVA DE LINEA TELEFONO DE CONTRATO

Se gestionarán las nuevas altas de línea para usuarios de contrato. Se recibirá a través del interfaz con el portal web una operación de alta con un tipo de línea de contrato.

- RF002 – ALTA NUEVA DE LINEA TELEFONO DE PREPAGO

Se gestionarán las nuevas altas de línea para usuarios de prepago. Se recibirá a través del interfaz con el portal web una operación de alta con un tipo de línea de prepago.

- RF003 – ALTA NUEVA LINEA ADICIONAL PARA UN TELEFONO PRINCIPAL

Se gestionarán las nuevas altas de líneas adicionales para una línea principal. Se recibirá una operación de alta de adicional indicando la línea principal y la línea adicional a dar de alta.

- RF004 – BAJA DE LINEA

Se gestionarán las bajas de línea que lleguen a nuestro sistema. Será necesario eliminar la información de la línea una vez recibamos esta operación.

- RF005 – CAMBIO DE LINEA

Se gestionarán las operaciones de cambio de línea que recibamos en nuestro sistema. Para la gestión de este tipo de operaciones se recibirán tanto la línea antigua como la nueva línea a crear.

- RF006 – ACTIVACION DE SERVICIO

Se gestionarán en nuestro sistema la activación de una serie de servicios que serán configurados por el cliente. De cara a nuestro sistema se recibirán la línea y el servicio que se quiere activar.

- RF007 – DESACTIVACION DE SERVICIO

Se gestionarán en nuestro sistema la desactivación de una serie de servicios que serán configurados por el cliente. De cara a nuestro sistema se recibirán la línea y el servicio que se quiere desactivar.

- RF008 – CONSULTA DATOS LINEA

El sistema deberá mostrar los datos correspondientes a las consultas que se hagan sobre las líneas. A nuestro sistema llegara la petición de consulta indicando la línea a consultar. Se deberá devolver tanto la línea dada de alta, la fecha de alta, el número de tarjeta sim asociada.

- RF009 – CONSULTA LINEAS ADICIONALES

El sistema deberá mostrar un listado de todas las líneas adicionales pertenecientes a una línea principal. A nuestro sistema llegará una petición en la que se indicará la línea principal a consultar. Se deberá devolver tanto el número de la línea adicional como la fecha en que fue dada de alta.

- RF010 – CONSULTA SERVICIOS

El sistema deberá mostrar un listado con los distintos servicios activados para un usuario de una línea. A nuestro sistema llegará una petición de consulta de servicios indicando la línea a consultar. Se deberá retornar tanto una listado con los distintos servicios activados para un usuario como la fecha de activación de los mismos.

2.3.2. REQUISITOS DE INTERFAZ

- RI001 – INTERFAZ OPERACIONES ALTAS,BAJAS,CAMBIOS DE LINEA Y GESTION DE SERVICIOS

Se deberá crear un nuevo interfaz para crear la comunicación entre la plataforma web y nuestro sistema. Dicho sistema estará conectado mediante protocolo TCP/IP y estará codificado en ASN.1. Deberá soportar todas las operaciones que se han comentado en la parte de requisitos funcionales y que nos deben llegar por este interfaz.

- RI002 – INTERFAZ TUXEDO OPERACIONES DE CONSULTA

Será necesaria la creación de un nuevo servidor tuxedo para la recepción de todas las posibles consultas que lleguen a nuestro sistema y que están descritas en el apartado de requisitos funcionales del sistema.

2.4. DIAGRAMA DE CASOS DE USO

El diagrama de casos de Uso consiste la descripción esquemática de todas las funcionalidades que va a tener un sistema, incluyendo los usuarios que van a poder realizar las distintas acciones. Este diagrama está compuesto de tres elementos:

- Actores: usuarios del sistema. Son las personas que accederán al sistema y pueden realizar las distintas acciones descritas en el punto de casos de uso. Para el caso que nos ocupa en nuestro sistema debido a que no se identifica el sistema y cada usuario puede realizar todas las operaciones accediendo al frontal web solo tendremos un único tipo de actor que es el cliente de nuestro sistema.
- Caso de Uso: cada una de las operaciones que se procesan en el sistema. En nuestro caso los casos de uso serán: altas de línea, bajas de línea, activación / desactivación de servicios, cambios de teléfono, consulta línea principal, consulta servicios y consulta de adicionales.
- Relaciones: Es el método de unión entre el actor y el caso de uso. Es la manera de especificar que operaciones pueden realizar cada actor del sistema.

2.4.1. DIAGRAMA DE CASOS DE USO

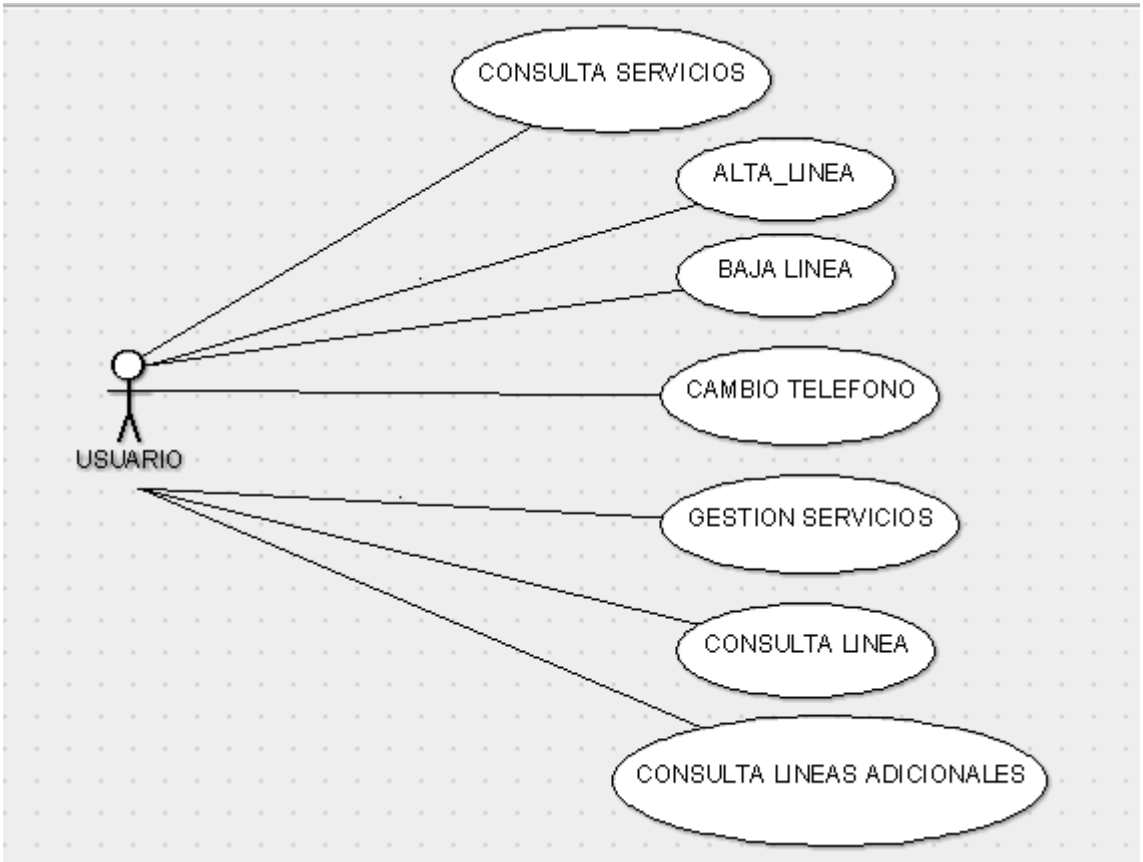


Ilustración 6: Diagrama de casos de uso

2.4.2. CATALOGO CASOS DE USO

Es el rol que se corresponderá con los usuarios del sistema.

Las acciones que podrá realizar serán:

2.4.2.1. ALTA DE LINEA

Caso de Uso:	*Alta de línea*
Código:	*CU.001*
Actores:	*Usuario web*
Descripción:	*Alta de línea de un nuevo usuario*
Origen:	**RF 001**
Precondiciones:	*Línea no dada de alta en el sistema.*
Postcondiciones:	*Línea dada de alta en el sistema*
Escenario principal	
1. Se recibe una petición de alta de línea desde la plataforma 2. Se realizan los chequeos pertinentes y se inserta el registro del nuevo usuario 3. Se responde a la plataforma dando el Ok.	
Escenarios secundarios	

2.4.2.2. BAJA DE LINEA

Caso de Uso:	*Baja de línea*
Código:	*CU.002*
Actores:	*Usuario web*
Descripción:	*Baja de línea de un usuario existente*
Origen:	**RF XXX**
Precondiciones:	*Línea de alta en el sistema*
Postcondiciones:	*La línea se da de baja en el sistema*

Escenario principal

1. Se recibe una petición de baja de línea desde la plataforma
2. Se realizan los chequeos pertinentes y se elimina el registro de la línea
3. Se responde a la plataforma dando el Ok.

Escenarios secundarios

2.4.2.3. CAMBIO DE TELEFONO

Caso de Uso:	*Cambio de teléfono*
Código:	*CU.003*
Actores:	*Usuario web*
Descripción:	*Cambio de línea*
Origen:	**RF XXX**
Precondiciones:	*Línea de alta en el sistema*
Postcondiciones:	*Cambio de la numeración de un usuario.*
Escenario principal	

1. Se recibe una petición de cambio de línea de un usuario.
2. Se realizan los chequeos pertinentes y se modifica el registro de la tabla con la nueva línea.
3. Se responde a la plataforma dando el Ok.

2.4.2.4. GESTION DE SERVICIOS

Escenarios secundarios	
Caso de Uso:	*Gestión de servicios*
Código:	*CU.004*
Actores:	*Usuario web*
Descripción:	*Activación / desactivación de un servicio de una línea dada de alta en el sistema*
Origen:	**RF XXX**
Precondiciones:	*Línea de alta en el sistema* - *Para la baja* *El servicio debe de estar activado para esa línea*
Postcondiciones:	*Se activa / desactiva el servicio solicitado para esa línea*
Escenario principal	
1. Se recibe una petición de activación / desactivación de un servicio para una línea. 2. Se realizan los chequeos pertinentes y se activa / desactiva el servicio dependiendo de la operación que llegue 3. Se responde a la plataforma dando el Ok.	
Escenarios secundarios	

2.4.2.5. CONSULTA DATOS LINEA

Caso de Uso:	*Consulta datos de línea*
Código:	*CU.005*
Actores:	*Usuario web*
Descripción:	*Consulta de los datos de una línea dada de alta*
Origen:	**RF 001**
Precondiciones:	*Línea dada de alta en el sistema.*
Postcondiciones:	
Escenario principal	
1. Se recibe una invocación al tuxedo de consulta de línea 2. Se realiza la consulta y se devuelve la información a la plataforma	
Escenarios secundarios	

2.4.2.6. CONSULTA LINEAS ADICIONALES

Caso de Uso:	*Consulta líneas adicionales*
Código:	*CU.006*
Actores:	*Usuario web*
Descripción:	*Consulta de las distintas líneas adicionales de una línea principal*
Origen:	**RF XXX**
Precondiciones:	*Línea dada de alta en el sistema.*
Postcondiciones:	

Escenario principal
1. Se recibe una invocación al tuxedo de consulta de líneas adicionales 2. Se realiza la consulta y se devuelve la información a la plataforma
Escenarios secundarios

2.4.2.7. CONSULTA SERVICIOS LINEA

Caso de Uso:	*Consulta servicios línea*
Código:	*CU.007*
Actores:	*Usuario web*
Descripción:	*Consulta de los distintos servicios dados de alta para un usuario*
Origen:	**RF XXX**
Precondiciones:	*Línea dada de alta en el sistema.*
Postcondiciones:	
Escenario principal	
1. Se recibe una invocación al tuxedo de consulta de servicios activados de una línea 2. Se realiza la consulta y se devuelve la información a la plataforma	
Escenarios secundarios	

2.5. SERVICIOS TUXEDO

Para la gestión de las consultas que se van a realizar en el sistema se van a implementar un servidor tuxedo que contendrá una serie de servicios que se van a corresponder con las distintas consultas que se pueden realizar.

1. Tx_consulta_datos_usuario: servicio tuxedo que realizará la consulta de los datos de un usuario. Como parámetro de entrada llegará el número de teléfono sobre el que se quieren consultar los datos. La salida estará compuesta por el número de teléfono, el número de la tarjeta sim asociada a la linea y la fecha de alta de la línea.
2. Tx_consulta_servicios: tuxedo que se encargará de obtener los distintos servicios que tiene activado un usuario. El parámetro de entrada del servicio será el número de teléfono de usuario. Este servicio tendrá como parámetros de salida los distintos servicios que tiene contratado el usuario y la fecha de altas de todos los servicios.
3. Tx_consulta_lineas_multisim: servicio tuxedo que se encargará de mostrar todas las líneas asociadas a una línea principal. Este servicio tuxedo tendrá como parámetro de entrada el número de teléfono del usuario. Los parámetros de salida serán las distintas líneas multisim contratadas y la fecha en que fueron dadas de alta las mismas.

Para la creación de estos servicios se deberá generar un servidor tuxedo al que hemos llamado Tx_consultas. Este servidor deberá ser creado en el ubbconfig y tendrá un idl que mostraremos más adelante.

2.6. PROCESOS BATCH

Para la gestión de las altas y bajas de líneas y de los servicios se van a crear una serie de procesos. Estas funcionalidades tendrá un patrón común, un proceso receptor (RECIBE) que es el encargado de la recepción de la tramitación y la inserción el la tabla de movimientos, un proceso gestor (GESTOR) que se encargará de tratar la petición y de los pertinentes chequeos y por último un proceso de envío (ENVIA) que se encargará de codificar la respuesta que se va a devolver y enviarla.

2.6.1. TRAMITACIONES DE ALTA

- **Proceso RECIBE_ALTA**

Proceso encargado de recibir una tramitación desde la plataforma web. Dicha tramitación llegará codificada en ASN.1 (el interfaz se mostrará posteriormente) y este proceso se encargará de la decodificación. Los parámetros que llegarán serán el número de teléfono, número de tarjeta sim, número de secuencia y tipo de línea (prepago, contrato o adicional). Posteriormente se insertará en la tabla de movimientos una nueva tramitación con los datos que hemos recibido añadiendo el código de operación dependiendo del tipo de línea que se desea dar de alta.

- **Proceso GESTOR_ALTA**

Proceso encargado de tramitar la petición. Este proceso estará leyendo de la tabla de movimientos y cuando llegue una nueva petición lo primero que hará será realizar un chequeo para comprobar si existe algún dato que no sea válido. A continuación, dependiendo de la operación que se vaya a realizar, se procederá al registro del nuevo usuario en el sistema. Por último se actualizará la tabla de movimientos para que la tramitación pase al módulo del envía.

- **Proceso ENVIA_ALTA**

Proceso que se encargará de finalizar la tramitación y responder a la plataforma web. Este módulo leerá la tramitación de la tabla de movimientos. Una vez leído eliminará la tramitación de esta tabla y la insertará en la tabla de históricos de movimientos. Se leerá el campo código de error y en caso de tramitación correcta se enviará respuesta OK a la plataforma. En caso de error se devolverá el código de error que se ha producido con una breve descripción.

2.6.2. TRAMITACIONES DE BAJA

- **Proceso RECIBE_BAJA**

Proceso encargado de recibir las tramitaciones de alta. Este proceso estará escuchando a un puesto determinado y en el momento que se reciba una solicitud de conexión se despertará y recibirá la tramitación que se debe ejecutar en ese momento. Dicha tramitación vendrá codificada en ASN.1 y tendrá como parámetros de entrada el número de teléfono, el tipo de línea, el número de tarjeta sim, el número de secuencia y una pequeña descripción de la causa de la baja. Una vez recibida la petición se insertará en la tabla de movimientos junto con el código de operación (que dependerá del tipo de línea que llegue) y se enviará una señal al gestor para que la tramite.

- **Proceso GESTOR_BAJA**

Proceso que se encarga de ejecutar la tramitación solicitada. Para ello lo primero que hará será realizar un cheque de que los parámetros son correctos: verificar que la línea está dada de alta, que es una línea sin restricciones, etc. Una vez se realiza este chequeo se pasara a la etapa de actualización que se encargará de dar de baja la línea en el sistema, la tarjeta sim asociada a la línea y dará de baja todos los servicios y líneas multisim que tuviera asociada la línea. Por ultimo enviará una señal al proceso envía para que recoja la tramitación.

- **Proceso ENVIA_BAJA**

Este proceso estará encargado de recoger la tramitación y enviar la respuesta a la plataforma. Para ello recogerá el movimiento en su estado final. Una vez leído el movimiento se encargará de montar la respuesta que se va a enviar a la plataforma para ello leerá el código de error que se ha registrado en el movimiento, se codificará a ASN.1 y se enviará a través del canal de conexión abierto anteriormente.

2.6.3. TRAMITACIONES DE GESTION DE SERVICIOS

- **Proceso RECIBE_SERVICIO**

Este proceso es el encargado de establecer la conexión con la plataforma. Una vez se haya establecido esta conexión se recibirá una petición de alta / baja de un servicio. Los parámetros que se recibirán son el teléfono, el número de secuencia, el servicio sobre el que se desea hacer una acción y la acción que se va a hacer (a: activar o d: desactivar). Se debe verificar si todos los parámetros vienen informados y en caso incorrecto de retornará un error a la plataforma. Una vez realizado este chequeo se generará en movimiento en la tabla de movimientos y se enviará una señal al gestor para que realice su tramitación.

- **Proceso GESTOR_SERVICIO**

Este proceso será el encargado de tramitar la petición del servicio. Lo primero que hará será leer el movimiento que llega. En función de la operación que llegue se harán unos chequeos, de tal forma que si llega una activación se deberá verificar que el teléfono está dado de alta, que el servicio que se quiere activar es compatible con el tipo de la línea sobre la que se quiere dar de alta y por último que ese servicio no está ya dado de alta. En caso de que llegue una desactivación se verificará únicamente que el servicio este dado de alta, en caso contrario se retornará un error. Una vez realizado el chequeo se actualizará el registro. Para el caso de las altas se insertara un nuevo registro en la tabla de servicios con el teléfono, el código de servicio, la fecha de alta y estado activado. Para el caso de las bajas se eliminará el registro de la tabla de servicios y se insertará en la tabla de históricos, en la que además de los campos

anteriormente descritos se incluirá un campo con la fecha de baja. Una vez realizado esto se actualizará el movimiento para que sea recogido por el proceso envía.

- **Proceso ENVIA_SERVICIO**

Proceso encargado de recoger las peticiones de altas o bajas de servicios y responder a la plataforma. Para ello leerá el movimiento recogiendo el código de error y el número de secuencia. Codificará esta información y se la retornará a la plataforma. Una vez realizado esto se eliminará el movimiento de la tabla de movimientos y se insertará el registro en la tabla de históricos.

2.6.4. TRAMITACIONES DE CAMBIO DE TELEFONO

- **Proceso RECIBE_CAMBIO_TLF**

Este proceso es el encargado de establecer la conexión con la plataforma. Una vez se haya establecido esta conexión se recibirá una petición de cambio de teléfono. Los parámetros que se recibirán son el teléfono antiguo, el nuevo número de línea, el número de la tarjeta sim asociada a la línea antigua y el número de secuencia. Se debe verificar si todos los parámetros vienen informados y en caso incorrecto de retornará un error a la plataforma. Una vez realizado este chequeo se generará en movimiento en la tabla de movimientos y se enviará una señal al gestor para que realice su tramitación.

- **Proceso GESTOR_CAMBIO_TLF**

Este proceso estará encargado de, en primer lugar hacer un chequeo para verificar que la línea antigua está dada de alta, que la nueva línea está en la tabla de teléfonos libres. Después de haber realizado se dará de baja la línea antigua de la tabla de teléfonos dados de alta sustituyéndola por la nueva línea. También será necesario sustituir la línea para todos los servicios que tuviera contratado anteriormente el usuario de esta línea. Una vez realizado esto se actualizará el movimiento para que sea recogido por el proceso envía.

- **Proceso ENVIA_CAMBIO_TLF**

Proceso encargado de recoger las peticiones de cambio de teléfono y responder a la plataforma. Para ello leerá el movimiento recogiendo el código de error y el número de secuencia. Codificará esta información y se la retornará a la plataforma. Una vez realizado esto se eliminará el movimiento de la tabla de movimientos y se insertará el registro en la tabla de históricos.

2.7. DISEÑO DE LA BASE DE DATOS

2.7.1. Introducción

En este apartado de la memoria vamos a detallar el diseño de la base de datos. Para ello crearemos el modelo relacional donde representaremos las entidades existentes en el sistema de provisión, así como sus atributos y relaciones. Así mismo identificaremos los distintos supuestos semánticos existentes en el modelo.

2.7.2. Esquema relacional

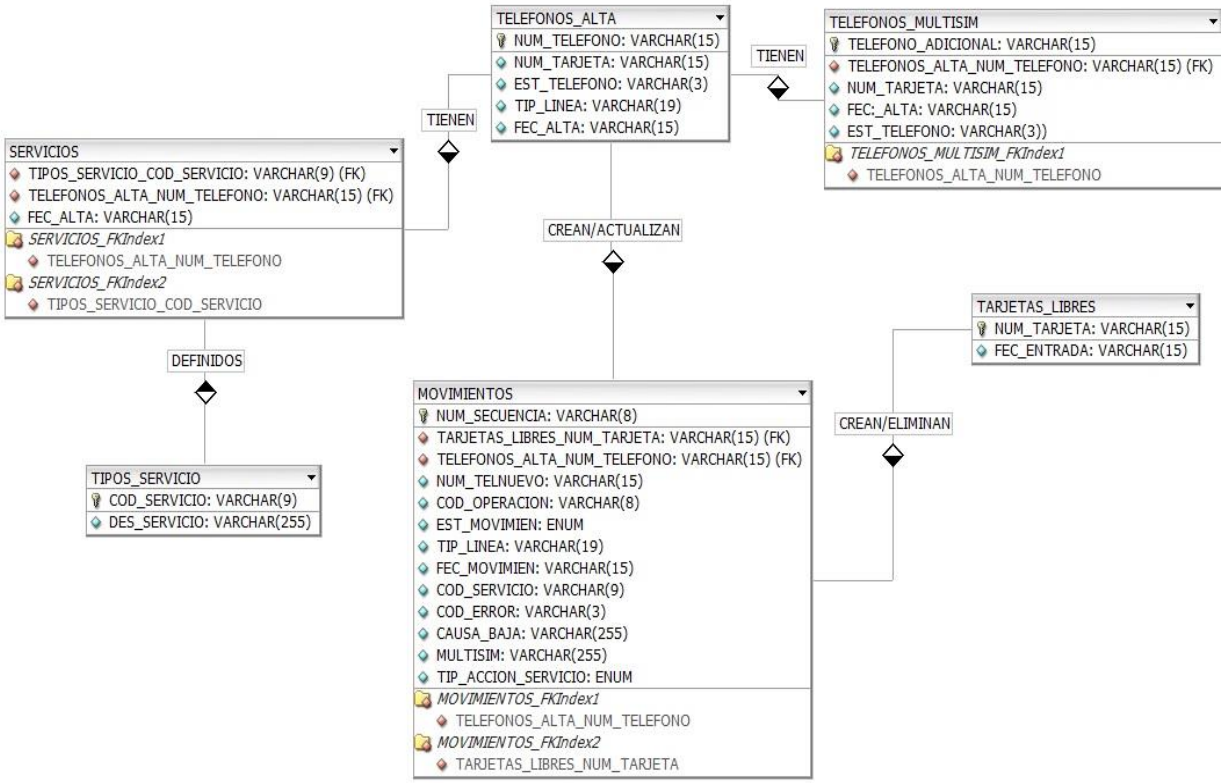


Ilustración 7: Esquema relacional

2.7.3. Modelo relacional

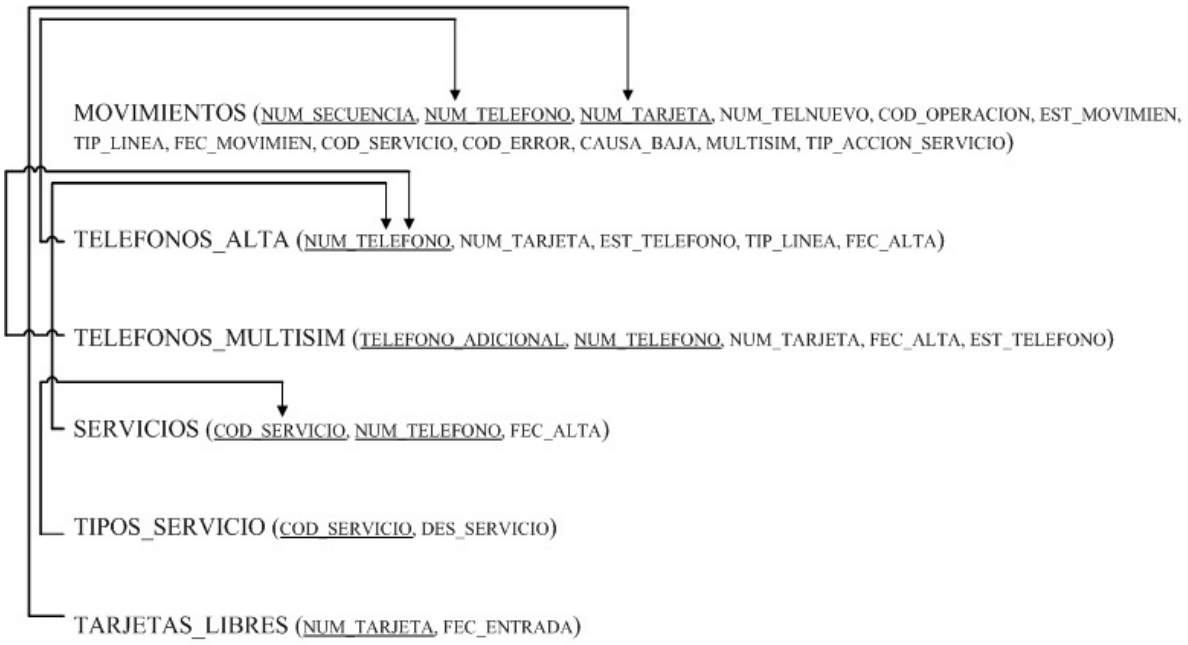


Ilustración 8: Modelo relacional

2.7.4. Supuestos semánticos

- **MOVIMIENTOS.EST_MOVIMIEN ϵ [“RECIBE”,”GESTOR”,”ENVIA”]**
- **MOVIMIENTOS.TIP_ACCION_SERVICIO ϵ [“A”,”D”]**
- **MOVIMIENTOS.TIP_LINEA ϵ [“CONTRATO”,”PREPAGO”]**
- **TELEFONOS_ALTA.TIP_LINEA ϵ [“CONTRATO”,”PREPAGO”]**
- **TELEFONOS_ALTA.EST_TELEFONO ϵ [“AE”,”BE”,”AS”,”BS”]**
- **TELEFONOS_MULTISIM.EST_TELEFONO ϵ [“AE”,”BE”,”AS”,”BS”]**

3. IMPLEMENTACIÓN DE LA APLICACIÓN

3.1. IMPLEMENTACIÓN POR CAPAS

Para el desarrollo de esta aplicación se ha tenido en cuenta la necesidad de facilitar el acceso a la información por parte de los usuarios. Por este motivo se ha decidido la inclusión de una capa de presentación que estará formado por los distintos interfaces que relacionarán nuestro sistema con el sistema gestor que nos realice las peticiones que tomará el rol de usuario de nuestra aplicación. Con el objetivo de facilitar la gestión de los datos que se van a tratar y posteriormente almacenar en la BBDD se ha optado por la inclusión de una capa de negocio. Por último contamos con una capa de datos en la que se engloba el acceso a base de datos. Por lo que contaremos con una arquitectura de tres capas que, aunque en uno de los casos no estará desarrollada por nosotros, si vemos oportuno incluirla en el desarrollo con el objetivo del entendimiento global del mismo.

1. Capa de presentación: en esta capa se va a explicar cómo vamos a interactuar con el usuario. Se explicarán los interfaces que se han creado para que el usuario de la aplicación (sistema gestor que se encuentre por encima de nuestra aplicación) pueda conectarse con nuestro sistema.
2. Capa de negocio: en esta capa es donde se especifica la funcionalidad de la aplicación en sí. Es la capa donde se realizan los desarrollos que permitan conectar las peticiones que llegan del usuario con las distintas actuaciones sobre la base de datos.
3. Capa de datos: es la capa donde se especifican todas las operaciones directas que se realizarán sobre la base de datos (INSERTS, DELETES, UPDATES, etc.). Al tratarse de una aplicación que gestionará un alto volumen de peticiones en un futuro será necesario una optimización máxima de acceso a la base de datos por lo que también se implementarán una serie de índices de búsqueda.

3.2. CAPA DE PRESENTACIÓN

En esta capa vamos a describir los distintos interfaces que van a ser entregados al sistema gestor que nos realice las peticiones. En nuestro caso vamos a dividir esta capa en dos partes, por un lado encontraremos el interfaz con los servicios tuxedo de la aplicación. En este apartado explicaremos tanto el idl que deberá pasarse al cliente para sus desarrollos y también veremos un ejemplo de llamada que se hará desde el cliente para cada una de las consultas ejecutadas. La otra parte de esta capa la ocuparemos para el desarrollo del interfaz ASN.1 para la llamada a los procesos batch que se van a encargar de ejecutar las tramitaciones que realizan la manipulación de datos dentro del sistema. En esta parte también se mostrarán unos ejemplos reales de peticiones que llegarán a nuestro sistema por parte del sistema gestor

3.2.1. INTERFAZ TUXEDO

Para este desarrollo hemos optado por la creación de un único servidor donde estarán encuadrados los tres servicios de consulta que hemos implementado. Para que el usuario pueda realizar las invocaciones a estos servicios se ha creado un idl que sirva de interfaz y que debe ser enviado al cliente para que lo incluya en sus desarrollo y pueda ejecutar las consultas.

El idl que se ha creado para esta funcionalidad tendrá la siguiente construcción:

```
Tx_consulta_datos_usuario, Tx_consulta_servicios y Tx_consulta_lineas_multisim. El
idl creado para esta funcionalidad tendrá el siguiente formato:

[
version(1.0)
]interface //Nombre del servidor
{

// Declaracion de constantes necesarioas
#define CONMGM_ESTADO_LEN 3
#define MSISDN_LEN 16
#define CONMGM_TIPOMSISDN_LEN 2
```

```
// Definicion de servicios
[
service_name(Tx_consulta_datos_usuario),
version(automatic)
] short Tx_consulta_datos_usuario(
    [in,string]      char Msisdn                [MSISDN_LEN],
    [out,string]    char Sim              [TARJETA_LEN],
    [out,string]    char FechaAlta        [FECHA_LEN]
    );
//RESULTADO DEL SERVICIO
//FINAL_OK (0): El servicio se ejecuta correctamente y devuelve los datos.
//FINAL_FAIL (X): Fallo en la ejecucion en el tuxedo

[
service_name(Tx_consulta_servicios),
version(automatic)
] short Tx_consulta_servicios(
    [in,string]      char Msisdn                            [MSISDN_LEN],
    [out,string]    char ArrayServicios                [TARJETA_LEN],
    [out,string]    char ArrayFechas                   [FECHA_LEN]
    );
//RESULTADO DEL SERVICIO
//FINAL_OK (0): El servicio se ejecuta correctamente y devuelve los datos.
//FINAL_FAIL (X): Fallo en la ejecucion en el tuxedo

[
service_name(Tx_consulta_lineas_multisim),
version(automatic)
] short Tx_consulta_lineas_multisim(
    [in,string]      char Msisdn                            [MSISDN_LEN],
    [out,string]    char ArrayMultisim           [TARJETA_LEN],
    [out,string]    char ArrayFechas                   [FECHA_LEN]
    );
//RESULTADO DEL SERVICIO
//FINAL_OK (0): El servicio se ejecuta correctamente y devuelve los datos.
//FINAL_FAIL (X): Fallo en la ejecucion en el tuxedo
}
```

En este documento vemos la declaración de los 3 servicios implementados (Tx_consulta_datos_usuario, Tx_consulta_servicios y Tx_consulta_lineas_multisim). En cada uno de estos servicios hemos identificado tanto los campos de entrada y de salida, así como el tipo de cada campo.

3.2.2. PROCESOS BATCH

Para la construcción del interfaz Frontal web – procesos BATCH se ha optado por el diseño de un interfaz ASN.1. Se ha creado un documento de interfaz que tiene la siguiente forma:

```
DEFINITIONS
    IMPLICIT TAGS :=
BEGIN
--**************DECLARAMOS LOS OBJETOS QUE VAMOS A UTIILIZAR***************
-- Alta de linea
altaLinea OBJECT-TYPE
SYNTAX altaLinea
ACCESS read-write
STATUS mandatory
::=1
-- Baja de linea
bajaLinea OBJECT-TYPE
SYNTAX BajaLinea
ACCESS read-write
STATUS mandatory
::=2
-- Gestion Servicios
gestionServicios OBJECT-TYPE
SYNTAX gestionServicios
ACCESS read-write
STATUS mandatory
::=3
```

```
--  Respuesta del Sistema de Provisión de Red
--  msgType     Type                Message
--  ============================================================
--  10      RespuestaError      respuestaError
--  11          RespuestaExito      respuestaExito
-- Respuesta de error a una petición
respuestaError OBJECT-TYPE
SYNTAX RespuestaError
ACCESS read-write
STATUS mandatory
::=10
-- Respuesta de confirmacion de recepcion de notificacion
respuestaExito OBJECT-TYPE
SYNTAX NULL
ACCESS read-write
STATUS mandatory
::=11
--============================================================


-- Alta de linea
altaLinea::= SEQUENCE
{
    num_telefono      [1] Msisdn,
    usuario            [2] ClaseCliente,
    DNI                [3] ClaseDni,
    tipo_linea        [4] ClaseLinea,
    direccion          [5] ClaseDireccion ,
    email             [6] ClaseEmail
};

-- Baja de linea
bajaLinea::= SEQUENCE
{
```

```
    num_telefono      [1] Msisdn,
    CausaBaja    [2] Descripcion,
};

-- gestion Servicios
gestionServicios ::= SEQUENCE
{
    num_telefono      [1] Msisdn,
    cod_servicio             [2] Servicio,
    acción            [3] TipoAccion,
};

-- Respuesta de error
RespuestaError ::= SEQUENCE
{
    codigo      [1] CodigoError
};

--Número de teléfono
Msisdn::=NumericString (SIZE(3..15));
--Datos usuario
ClaseCliente::=PrintableString (SIZE(10..250)):
--Clase Dni (por definir)
ClaseDni::=PrintableString (SIZE(9));
--Clase de línea a dar de alta: contrato o prepago
ClaseLinea::= PrintableString (SIZE(3)):
--Direccion del usuario
ClaseDireccion::= PrintableString (SIZE(10..250)):
--Email del usuario
ClaseEmail ::= PrintableString (SIZE(6..120)):
--Causa de la baja
Descripcion::= PrintableString (SIZE(10..250)):
--Servicios que se pueden activar/desactivar
```

```
Servicio::= PrintableString (SIZE(6)):
--Accion con el servicio: AC (activar) o DE (desactivar)
TipoAccion::= PrintableString (SIZE(2)):
--Errores a devolver
CodigoError ::= INTEGER {
    buzonNoExiste(1),
        -- Este error lo genera la petición de consulta de buzón si no existe el
        -- buzón. El resto de operaciones crean automáticamente el buzón.
    operacionNoPermitida(2),
        -- El perfil de cliente no permite la operación solicita. En principio
        -- este error se produce al pedir un alta de fax, pero puede que el
        -- resto de operaciones llegen a generarlo algún día.
    errorEnParametros(3),
        -- La operación se ha rechazado por un error en la validación de
        -- los parámetros.
    errorTemporal(4),
        -- La operación no se puede ejecutar por un error temporal en el
        -- sistema. Debería reintentarse trascurrido cierto tiempo.
    errorInterno(5),
        -- Error interno en el sistema de gestión automática de buzones.
        -- No reintentar la operación, contactar con el soporte de SGAB.
    errorOperador(6)
        -- Este error lo genera cualquier peticion que lleve el parametro
        -- "proveedor" si el dato que se envia no coincide con el que hay
        -- registrado en SGAB para el buzon
};
```

En este documento podemos identificar 3 partes:

- En una primera parte declaramos los objetos que pueden intervenir en el intercambio, el objeto correspondiente con el alta de línea (altaLinea), el de baja de línea (bajaLinea) y el de gestión de servicios (gestionServicios). Así

mismo declararemos los códigos de error y éxito que devolveremos en la respuesta.

- En la segunda parte declaramos las estructuras asociadas a los objetos. Por cada objeto asociamos los campos que van a viajar.
- En la última parte nos encargamos de identificar los distintos campos pertenecientes a las estructuras identificando el tipo de campo y la longitud del mismo

3.3. CAPA DE NEGOCIO

En este apartado vamos a desarrollar el proceso seguido para el desarrollo de todas las funcionalidades que va a poseer nuestro sistema. Empezaremos por la parte de tuxedo de la aplicación donde daremos una indicación de cómo hemos creado los servicios e indicaremos el flujo que seguirán cada una de estas funcionalidades. En la segunda parte explicaremos el desarrollo de los procesos batch que se han creado.

3.3.1. SERVIDOR TUXEDO

Como hemos indicado anteriormente nuestro servidor de consultas estará compuesto por tres servicios: uno de consulta de línea, otro de consulta de servicios y otro para consultar las líneas adicionales que tiene una línea principal. Para la realización de nuestras pruebas nos hemos ayudado de un cliente de prueba, un cliente_tux, que nos va a hacer las labores de la parte cliente de nuestra aplicación.

3.3.1.1. CONSULTA DATOS LINEA

Para esta prueba se ha usado un cliente_tux que nos simulará las peticiones que recibiremos desde la parte cliente. La invocación realizada para estas pruebas será la siguiente:

```
cliente_tux 28458512 Tx_consulta_datos_usuario is “617938060” os os
```

Dónde:

- cliente_tux: binario de ejecución del cliente tux.
- 28458512: base generada de la compilación del idl.
- Tx_consulta_datos_usuario: nombre del tuxedo a ejecutar.
- is “617938060”: indica que se trata de un parámetro de entrada de tipo string y se indica el valor del mismo.
- os: parámetro de salida de tipo string. Es necesario incluir tantos como parámetros de salida haya.

Una vez ejecutado el cliente se realizará una llamada a nuestro servidor que tendrá la siguiente operativa:

- En primer lugar se ejecutará una Select sobre la tabla TELEFONOS_ALTA sobre la que se consultará. Para esta funcionalidad se ha creado un cursor que se encargará de recuperar el registro localizado.

```
SQL> DESC TELEFONOS_ALTA
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
  NUM_TELEFONO                              NOT NULL VARCHAR2(15)
 EST_TELEFONO                              NOT NULL VARCHAR2(2)
 NUM_ICC                                   NOT NULL VARCHAR2(19)
 FEC_ALTA                                  NOT NULL DATE
  TIPO_LINEA                                NOT NULL CHAR(1)


SQL> select * from TELEFONOS_ALTA where num_telefono='617938060';

617938060       AE 555555555555555
2013/10/23 17:51:44    C

SQL>
~
~
```

Ilustración 9: Consulta tuxedo linea (I)

Una vez se recuperen los datos de la consulta se retornarán a la salida del servicio tuxedo en los parámetros reservados para la salida.

```
9 13:56:19 [] El numero de telefono a consultar es : 617938060
9 13:56:19 [] CLIENTE - Tx_consulta_datos_usuario - Parametros de entrada:
9 13:56:19 []    Msisdn ......: 617938060
9 13:56:19 [] CLIENTE - Tx_consulta_datos_usuario - Parametros de salida:
9 13:56:19 []    Tarjeta_Sim .: 555555555555555
9 13:56:19 []    Fecha_Alta .........: 20131023175144
9 13:56:19 [] fin_transaccion= 1, timeout=10000
9 13:56:20 [] SERVIDOR - Tx_consulta_datos_usuario - Resultado: 0
```

Ilustración 10: Consulta tuxedo linea (II)

3.3.1.2. CONSULTA DE SERVICIOS

Para realizar la prueba de este servicio utilizaremos de nuevo el cliente tuxedo anterior. Deberemos cambiar el nombre del servicio por el de consulta de servicios.

Nuestro servicio en primer lugar se encargará de realizar una consulta sobre la tabla TELEFONOS_ALTA para verificar que el teléfono se encuentra dado de alta en sistema. En caso de que no se encuentre se retornará un error al cliente:

```
SQL> select * from TELEFONOS_ALTA where num_telefono='617938060';
617938060       AE 555555555555555
2013/10/23 17:51:44     C
```

El siguiente paso que hacemos en caso de que nuestra línea se encuentre dada de alta será el de crear un cursor para que se recuperen los datos de los servicios dados de alta para el usuario. Se creará un bucle que recorra la tabla SERVICIOS recuperando todos los servicios que estén dados de alta para la línea a consultar:

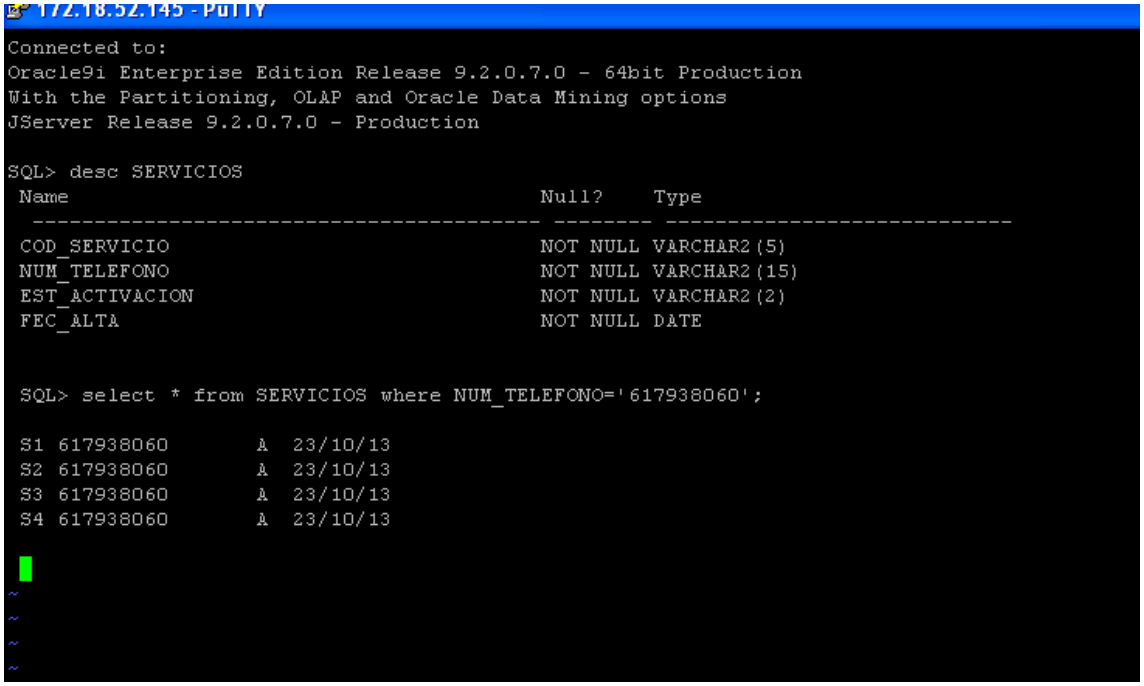


Ilustración 11: Consulta tuxedo servicio (I)

Una vez vayamos recuperando los datos de la consulta iremos concatenando los valores de los servicios y las fechas de alta en los dos arrays de salida que estarán separados por "$". Posteriormente se responderá a la plataforma con estos valores:

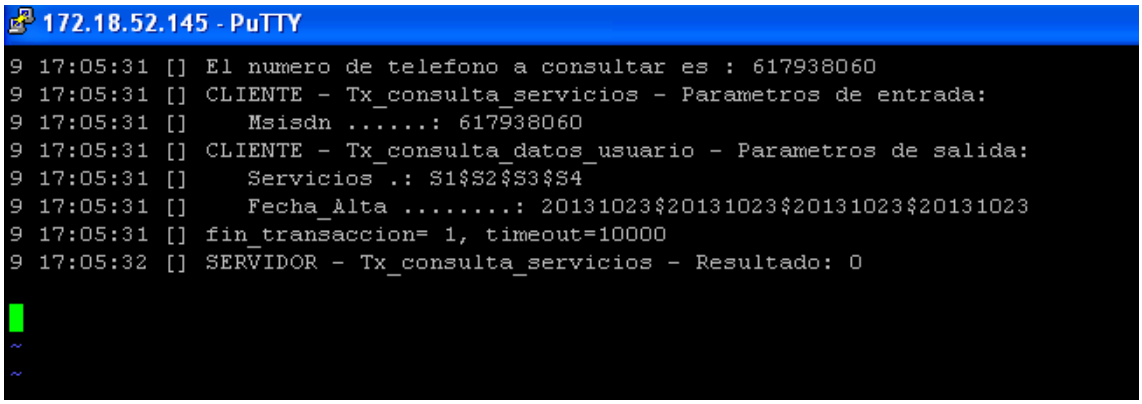


Ilustración 12: Consulta tuxedo servicio (II)

3.3.1.3. CONSULTA LINEAS ADICIONALES

El último servicio de consulta incluido en nuestro desarrollo es de la consulta de las líneas adicionales de una línea principal. Para esta nueva consulta una vez más ejecutaremos el cliente tux cambiando el nombre del servicio por el que vamos a ejecutar ahora (Tx_consulta_lineas_multisim).

Nos llegará una invocación a este servicio que tendrá como parámetro de entrada la línea a consultar. Lo primero que haremos será verificar si la línea principal se encuentra dada de alta en el sistema. En caso de que no estuviera se retornaría un error al sistema usuario.

```
SQL> select * from TELEFONOS_ALTA where num_telefono='617938060';
617938060       AE 555555555555555
2013/10/23 17:51:44     C
```

Una vez realizada la consulta inicializaremos un cursor que dentro de un bucle irá leyendo todos los registros de la tabla TELEFONOS_ADICIONALES cuyo campo número_principal coincida con el de la consulta. Cada vez que encontremos un valor concatenaremos en los arrays de salida tanto el valor de la línea adicional (campo número_adicional) como la fecha en que fue dado de alta en el sistema.

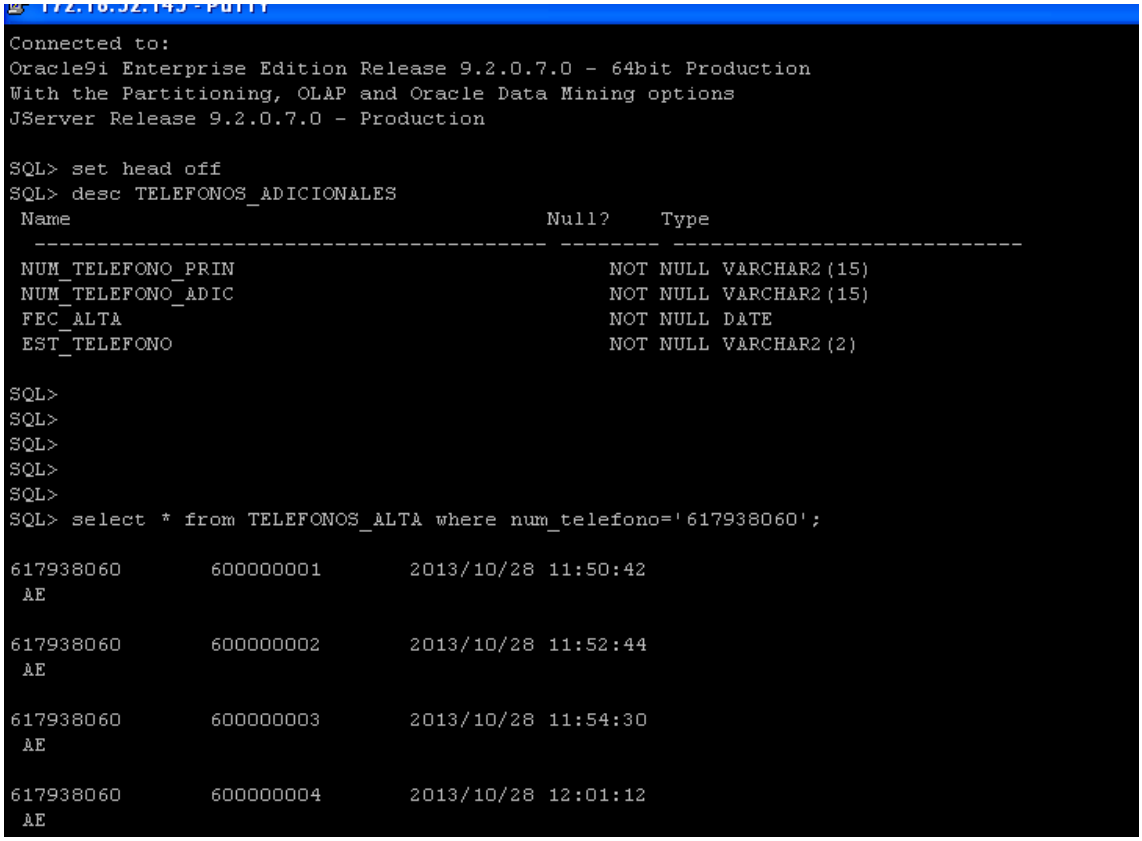


Ilustración 13: Consulta tuxedo adicionales (I)

Una vez finalizada la búsqueda eliminamos el cursor y retornamos los parámetros de salida a la plataforma:

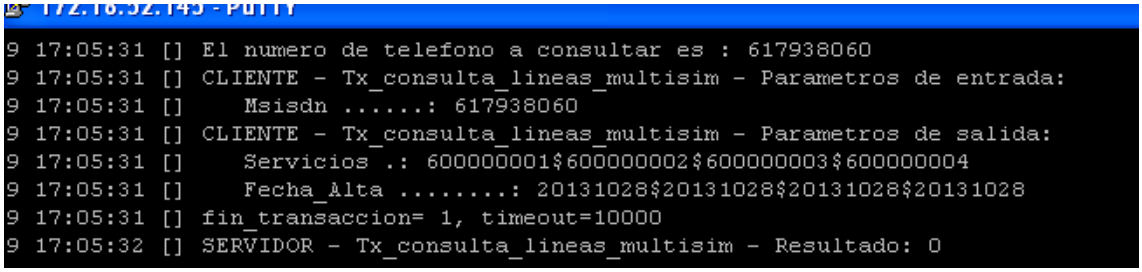


Ilustración 14: Consulta tuxedo adicionales (II)

3.3.2.PROCESOS BATCH

Como segunda parte de la aplicación hemos desarrollado una serie de procesos batch para todas aquellas tramitaciones que no van a exigir una respuesta online al sistema gestor. Vamos a explicar su funcionamiento a través de una serie de operaciones realizadas sobre estos procesos. Las operaciones que incluimos son: alta de línea, baja de línea, gestión de servicios, cambio de línea y alta y baja de líneas multisim. Vamos a ir indicando paso por paso la gestión que hace el sistema en cada paso de la operación y mostrando la evolución de la base de datos.

3.3.2.1. PROCEDIMIENTO ALTA DE LINEA

Cuando desde el sistema gestor nos llegue una solicitud de alta de línea lo primero que haremos es leer la trama ASN1 que nos llegue y evaluar si se trata de una operación de alta de línea normal o un alta de multisim. Para ello nos fijaremos si el campo tipoAlta viene informado en la trama. El proceso que se encarga de esta recepción (RECIBE_ALTA) extrae la trama ASN1 y lo decodifica mediante las funciones obtenidas (BDecAltaLinea) una vez ejecutada la compilación del interfaz ASN1 acordado, recogido anteriormente.

Los valores a insertar en la tabla de movimientos son:

NOMBRE DEL CAMPO	VALOR
NUM_SECUENCIA	Valor del campo numSecuencia que llega en la trama ASN.1
NUM_TELEFONO	Valor del campo número que llega en la trama ASN.1
NUM_TARJETA	Valor del campo TarjetaSim que llega en la trama ASN.1
TIP_LINEA	Valor del campo TipoLinea que llega en la trama ASN.1
COD_OPERACION	En caso de alta normal valor “ALTA_NORMAL” en caso de línea multisim “ALTA_MULTISIM”
MULTISIM	Valor del campo Lineas_adicionales que llega en la trama ASN.1

El ASN1 que recibiremos desde el sistema gestor tendrá el siguiente formato:

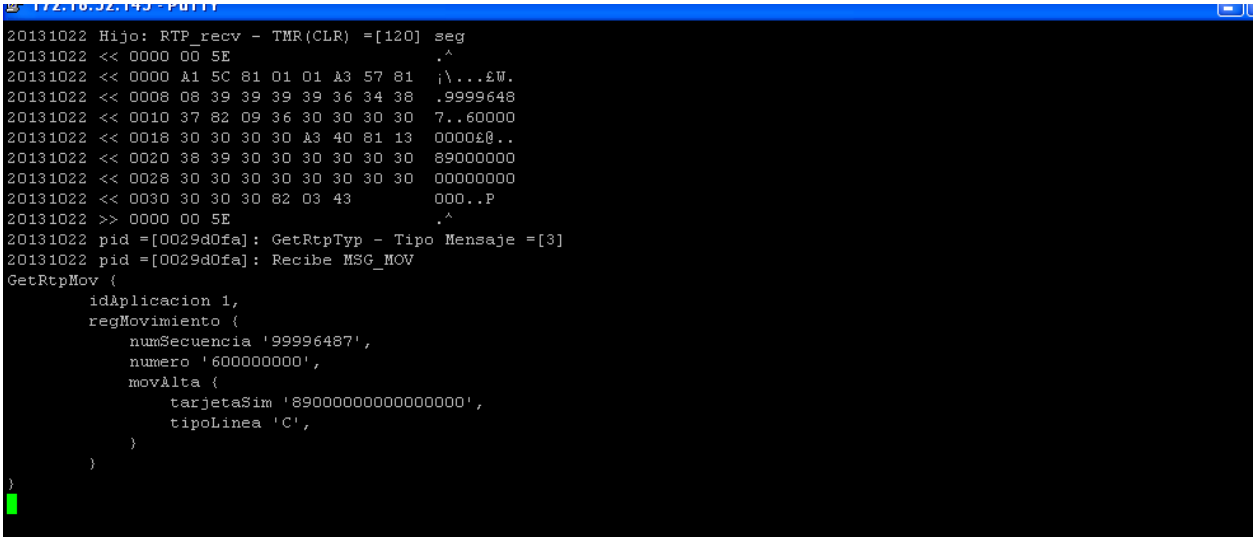


Ilustración 15: Ejemplo alta de línea (I)

Una vez recibida la petición se procede a ejecutar el INSERT sobre la tabla de movimientos (Insert_MOVIMIENTO), que insertará una tramitación para que la recoja el proceso gestor. El insert que se ejecuta será mostrado posteriormente en la parte de anexos:

El estado de la BBDD será el siguiente:

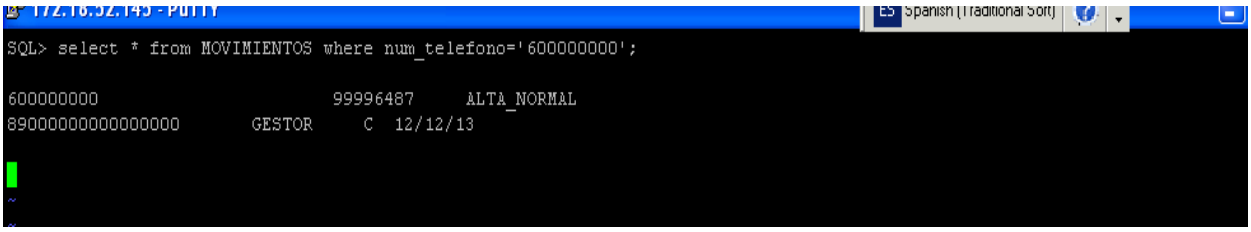


Ilustración 16:Ejemplo alta de línea (II)

Una vez que se inserte el registro con el campo EST_MOVIMIENTO=’GESTOR’ el control de la tramitación será tomado por el proceso GESTOR_ALTA. En este modulo es

donde se ubica la gran parte de la funcionalidad del sistema. Este proceso realizará una serie de verificaciones para validar el alta:

- Se buscará la línea en la tabla TELEFONOS_LIBRES (Select_EST_TELEFONO_LIBRE) que verificará que la línea introducida se encuentra disponible en el sistema.
- La siguiente cosa a verificar es que la tarjeta introcida también se encuentra dispoble. Para ello se buscará en la tabla TARJETAS_LIBRES (Select_EST_TARJETA_LIBRE).

Una vez pasado estos chequeos procederemos a dar de alta la línea en el sistema. Para ello lo primero que haremos será crear un registro en la tabla TELEFONOS_ALTA (Insert_EST_TELEFONO) con los datos tanto de la línea como de la tarjeta asociada a la misma. Como la numeración será ocupada será necesario tanto eliminar el registro de la tabla TELEFONOS_LIBRES (Delete_TELEFONO_LIBRE) como de la TARJETAS_LIBRES (Delete_TARJETA_LIBRE).

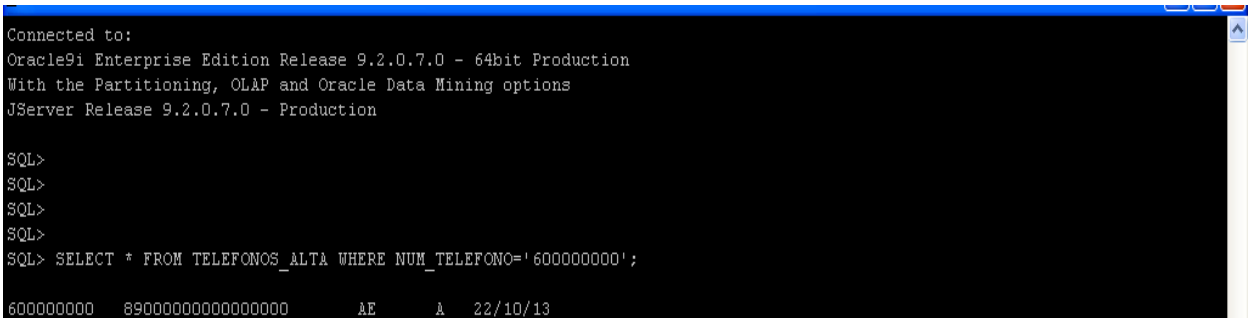


Ilustración 17: Ejemplo alta de línea (III)

Una vez pasado este proceso el gestor pasará el control al proceso ENVIA_ALTA que se encargará de responder al sistema gestor.

Este proceso se encargar de recoger la tramitación de la tabla de movimientos. Verifica el campo sCOD_ERROR que será nulo en caso de que no haya habido fallo. Codificará el ASN1 resultante y posteriormente procederá a enviarlo. Una vez recibamos la señal de ÉXITO por parte del sistema gestor eliminaremos el registro de la tabla de tramitaciones (int Delete_MOVIMIENTO (movimiento *pt).

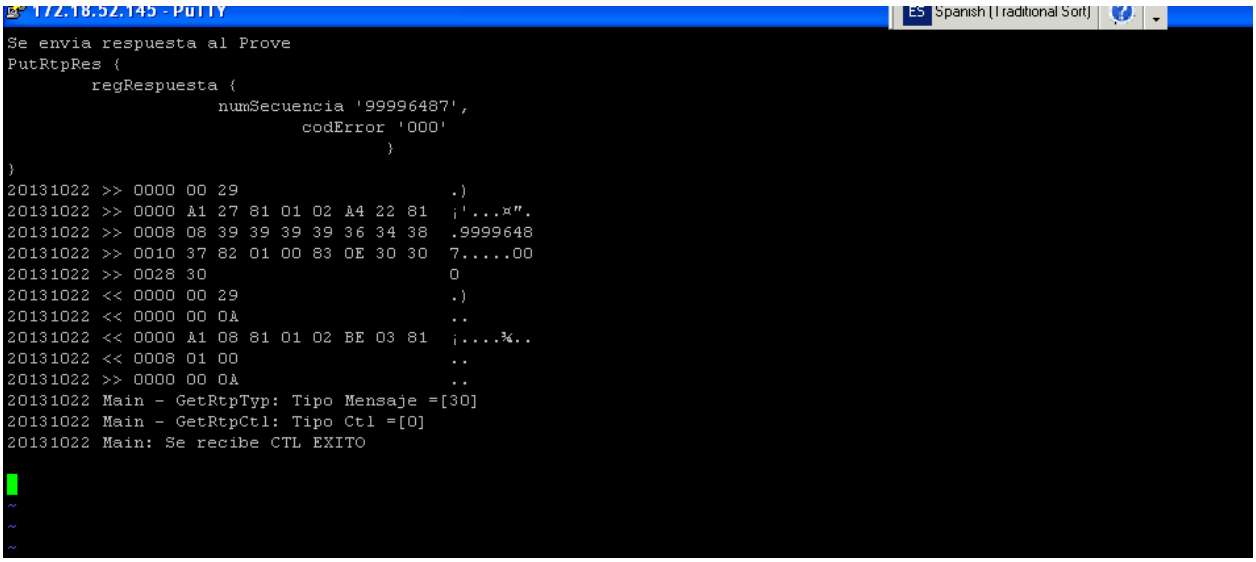


Ilustración 18: Ejemplo alta de línea (IV)

3.1.1.1. PROCEDIMIENTO BAJA DE LINEA

En el momento que nos llegue una tramitación de baja para una línea primeramente nos encargaremos de decodificar la trama ASN.1 que recibamos desde el sistema gestor. Una vez decodificado crearemos la tramitación en la tabla de movimientos para formar la baja. Los campos a insertar en esta tabla son los siguientes:

NOMBRE DEL CAMPO	VALOR
NUM_SECUENCIA	Valor del campo numSecuencia que llega en la trama ASN.1
NUM_TELEFONO	Valor del campo número que llega en la trama ASN.1
DES_CAUSA	Valor del campo CausaBaja que llega en la trama ASN.1
COD_OPERACION	"BAJA"
EST_MOVIMIEN	"GESTOR"

Ahora se muestra como llegaría la trama ASN.1 y su decodificación:

```
20131024 Hijo: RTP_recv - TMR(CLR) =[120] seg
20131024 << 0000 00 5E                    .^
20131024 << 0000 A1 5C 81 01 01 A3 57 81  ¡\...£W.
20131024 << 0008 08 39 39 39 39 36 34 38  .9999648
20131024 << 0010 37 82 09 36 30 30 30 30  7..60000
20131024 << 0018 30 30 30 30 A3 40 81 13  0000£@..
20131024 << 0020 50 65 74 69 63 69 6F 6E  Peticion
20131024 << 0028 20 55 73 75 61 72 69 6F   Usuario
20131024 >> 0000 00 5E                    .^
20131024 pid =[0029d0fa]: GetRtpTyp - Tipo Mensaje =[3]
20131024 pid =[0029d0fa]: Recibe MSG_MOV
GetRtpMov {
        idAplicacion 1,
        regMovimiento {
            numSecuencia '99996487',
            numero '600000000',
            movBaja {
                CausaBaja  'Peticion Usuario',
            }
        }
}
```

Ilustración 19: Ejemplo baja de línea (I)

Una vez hayamos insertado la tramitación de baja en la tabla de movimientos empezaremos con el chequeo de la petición. Para verificar la validez de la tramitación se verificará lo siguiente:

- Se validará que el teléfono se encuentra en la tabla (Select_EST_TELEFONO) que pasando como parámetro una estructura de tipo teléfono con el valor del número buscará en la tabla TELEFONOS_ALTA el teléfono. En caso de no encontrarlo retornará un error controlado.

Cuando ya se haya realizado este chequeo se realizará las actualizaciones sobre BBDD. En primer lugar insertaremos el teléfono y la tarjeta a liberar en las tablas TELEFONOS_LIBRES (Insert_EST_TELEFONO_LIBRE) y TARJETAS_LIBRES (Insert_EST_TARJETA_LIBRE) para volver a usarlas. Por último eliminaremos el registro de la tabla TELEFONOS_ALTA (Delete_TELEFONO_ALTA).

El estado de la BBDD en este momento será:

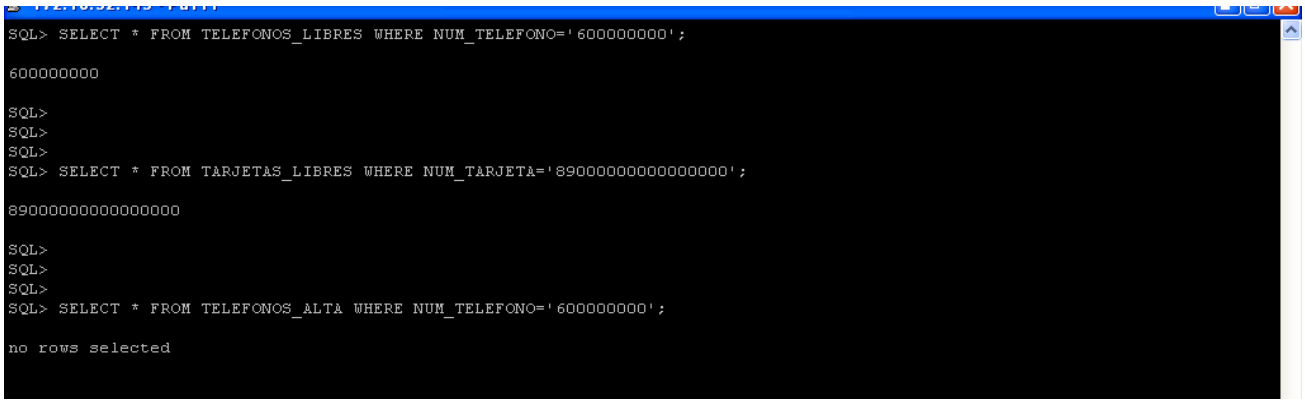


Ilustración 20: Ejemplo baja de línea (II)

Una vez realizadas estas operaciones actualizaremos la tramitación (EST_MOVIMIEN=’ENVIA’) para pasar el control del movimiento al proceso envía que se encargará de mandar la respuesta al sistema gestor:

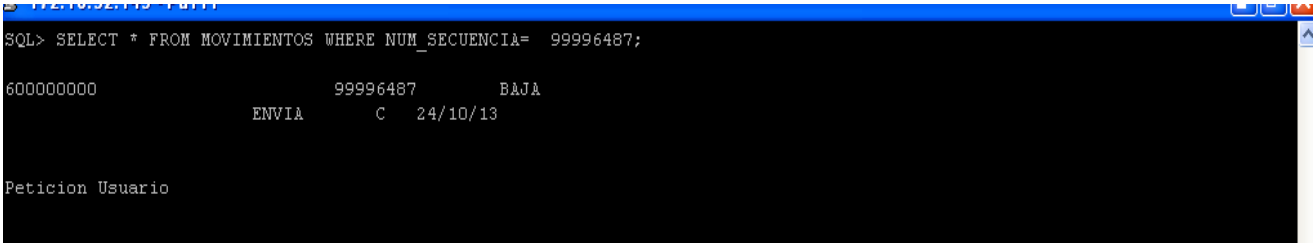


Ilustración 21: Ejemplo baja de línea (III)

Por último de la misma manera que hemos visto antes con el proceso ENVIA de las operaciones de ALTA se enviará la respuesta a la plataforma indicando la correcta realización de la tramitación.

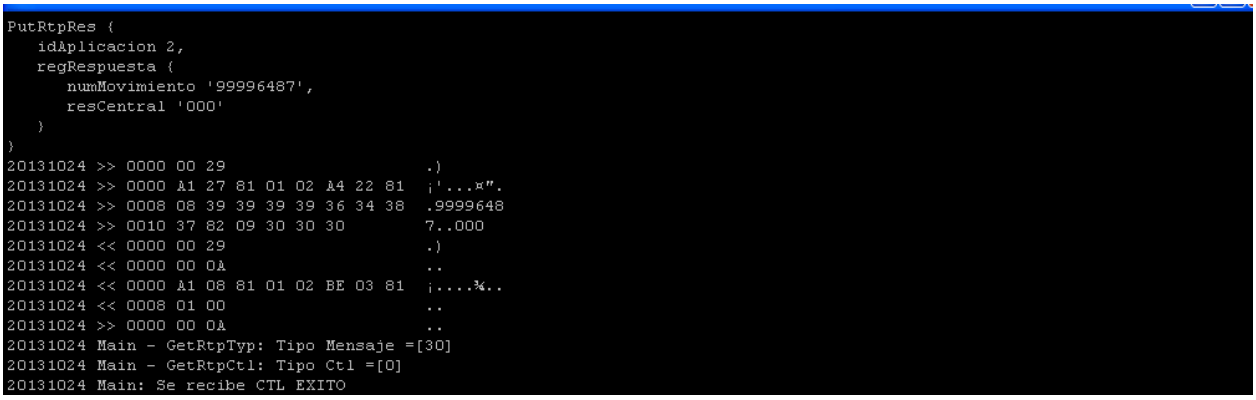


Ilustración 22: Ejemplo baja de línea (IV)

3.3.2.2. ALTA/BAJA DE SERVICIOS

Primeramente recibiremos una petición con origen el sistema gestor en la que se nos especificará tanto la línea sobre la que se va a realizar la acción, el servicio que se quiere gestionar y la acción que se va a realizar (ACTIVAR/DESACTIVAR).

NOMBRE DEL CAMPO	VALOR
NUM_SECUENCIA	Valor del campo numSecuencia que llega en la trama ASN.1
NUM_TELEFONO	Valor del campo número que llega en la trama ASN.1
COD_SERVICIO	Valor del campo `cod_servicio` que llega en la trama ASN.1
cTIP_ACCION_SERVICIO	Tomará los valores: A -> ACTIVACION D -> DESACTIVACION

Esta sería la trama ASN.1 que se recibe y su decodificación correspondiente:

```
20140121 Hijo: RTP_recv - TMR(CLR) =[120] seg
20140121 << 0000 00 5E                    .^
20140121 << 0000 A1 5C 81 01 01 A3 57 81  ¡\...£W.
20140121 << 0008 08 39 39 39 39 36 34 38  .9999648
20140121 << 0010 38 82 09 36 30 30 30 30  8..60000
20140121 << 0018 30 30 30 30 A3 40 81 13  0000£@..
20140121 << 0020 53 65 74 76 69 63 69 6F  Servicio
20140121 << 0028 31                       1
20140121 >> 0000 00 5E                    .^
20140121 pid =[0031e5db]: GetRtpTyp - Tipo Mensaje =[3]
20140121 pid =[0031e5db]: Recibe MSG_MOV
GetRtpMov {
    idAplicacion 1,
    regMovimiento {
        numSecuencia '99996488',
        numero '600000000',
      movServicio {
         cod_servicio 'Servicio1',
         accion 'A'
        }
    }
}
```

Ilustración 23: Ejemplo alta/baja servicios (I)

Una vez creada la tramitación cambiamos al estado de “GESTOR” donde primeramente se realizarán una serie de chequeos:

- Se verificará que la línea a gestionar este de alta en la tabla TELEFONOS_ALTA.
- Se comprueba que el servicio no está ya de alta para la línea (Select_EST_SERVICIO).

Después de haber verificado esto pasaremos a actualizar la tabla de servicios (SERVICIOS). Para ello llamados a la función Insert_EST_SERVICIO que insertará un nuevo registro en la tabla SERVICIOS con los siguientes valores:

NOMBRE DEL CAMPO	VALOR
NUM_TELEFONO	Número de teléfono que desea dar de alta el servicio
COD_SERVICIO	Servicio a dar de alta
EST_SERVICIO	Posibles valores: A -> Activado D -> Desactivado
FEC_ALTA	Fecha en la que llega la tramitación (sysdate)

El estado de esta tabla una vez ejecutado el insert será el siguiente:

```
Copyright (c) 1982, 2007, Oracle.  All Rights Reserved.


Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.2.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
SQL>
SQL>
SQL>
SQL> select * from SERVICIOS where NUM_TELEFONO='600000000';

NUM_TELEFONO    COD_SERVI ES FEC_OPERACIO
--------------- --------- -- ----------------
636960502       Servicio1 A  21/01/2014 12:33
```

Ilustración 24: Ejemplo alta/baja servicios (II)

Por último se pasaría el control de la tramitación al proceso ENVIA (EST_MOVIMIEN='ENVIA' en la tabla MOVIMIENTOS). Este proceso recogería la tramitación codificaría la respuesta en un ASN.1 y realizaría el envío de la respuesta al sistema gestor.

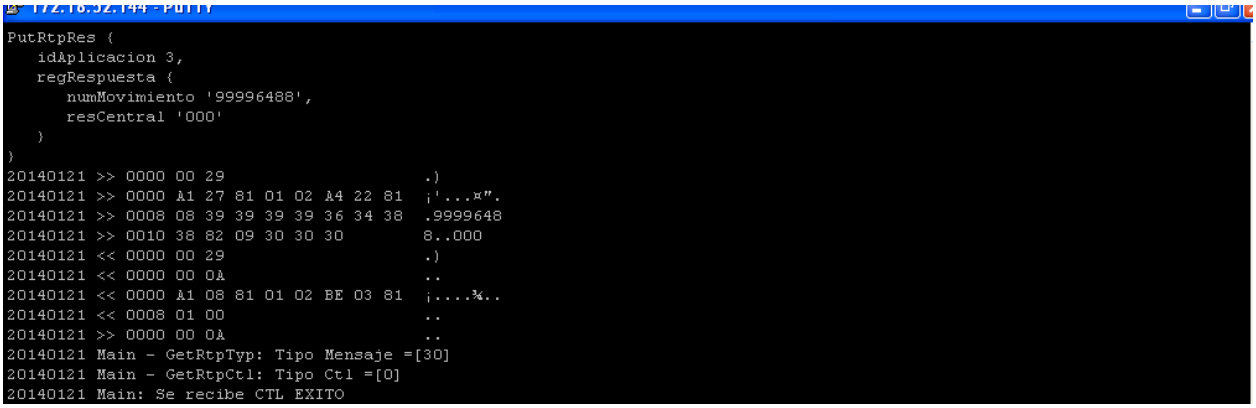


Ilustración 25: Ejemplo alta/baja servicios (III)

Para el caso de operaciones de baja de servicios existen unas pequeñas diferencias con respecto a las altas:

- En el "RECIBE" la única diferencia es que en la operación nos llegada una "D" (Desactivación).
- En el módulo "GESTOR" se chequeará lo mismo salvo que en caso de que servicio ya este dado de alta se elimina. En caso de que no esté de alta se retornará un error indicándolo.
- El módulo "ENVIO" no sufrirá ninguna modificación.

3.3.2.3. CAMBIO DE TELEFONO

En las operaciones de cambio de teléfono los únicos parámetros que recibiremos serán, a parte del número de secuencia, el teléfono antiguo y el nuevo teléfono. Se procederá una vez más a decodificar el ASN.1 y se realizará el Insert sobre la tabla de movimientos. Los parámetros a insertar son:

NOMBRE DEL CAMPO	VALOR
NUM_SECUENCIA	Valor del campo numSecuencia que llega en la trama ASN.1
NUM_TELEFONO	Valor del campo numeroViejo que llega en la trama ASN.1
NUM_TELNUEVO	Valor del campo numeroNuevo que llega en la trama ASN.1

El ASN.1 recibido tendrá el siguiente formato:

```
20140122 Hijo: RTP_recv - TMR(CLR) =[120] seg
20140122 << 0000 00 5E                    .^
20140122 << 0000 A1 5C 81 01 01 A3 57 81  ¡\...£W.
20140122 << 0008 08 39 39 39 39 36 34 38  .9999648
20140122 << 0010 39 82 09 36 30 30 30 30  9..60000
20140122 << 0018 30 30 30 30 A3 40 81 13  0000£@..
20140122 << 0020 36 30 30 30 30 30 30 30  60000000
20140122 << 0028 31                       1
20140122 >> 0000 00 5E                    .^
20140122 pid =[0021c1ge]: GetRtpTyp - Tipo Mensaje =[3]
20140122 pid =[0021c1ge]: Recibe MSG_MOV
GetRtpMov {
        idAplicacion 4,
        regMovimiento {
            numSecuencia '99996487',
            numeroViejo '600000000',
            numeroNuevo '600000001'
        }
}
```

Ilustración 26: Ejemplo cambio teléfono (I)

La tramitación pasará a manos del gestor que se encargará de realizar una serie de chequeos y las modificaciones sobre BBDD. Los chequeos que va a realizar son:

- Se verificará que la línea antigua que llega (campo NUM_TELEFONO de la tabla MOVIMIENTOS) se encuentra actualmente de alta (en la tabla TELEFONOS_ALTA).
- Validar que la nueva línea (campo NUM_TELNUEVO de la tabla MOVIMIENTOS) se encuentra actualmente como numeración libre (registro en TELEFONOS_LIBRES)

Una vez verificados estos términos se realizarán las modificaciones sobre la base de datos. En primer lugar crearemos un registro en la tabla de teléfonos libres (TELEFONOS_LIBRES) con la numeración que se va a liberar (campo NUM_TELEFONO de la tabla MOVIMIENTOS). Seguidamente actualizaremos la tabla de líneas de alta indicando el valor del nuevo teléfono (campo NUM_TELNUEVO de la tabla MOVIMIENTOS). Para realizar la actualización se ejecutará la función Update_EST_TELEFONO. Una vez realizadas estas modificaciones la base de datos tendrá un estado similar al siguiente:

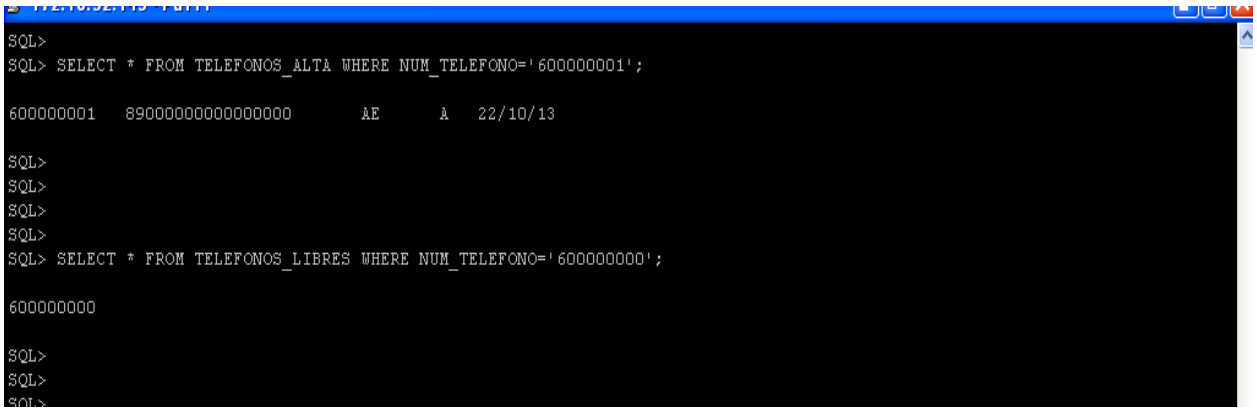


Ilustración 27: Ejemplo cambio teléfono (II)

Por último el control de la tramitación lo cogerá el modulo “ENVIA” que se encargará de enviar la respuesta al sistema gestor para informarle de la finalización de la tramitación.

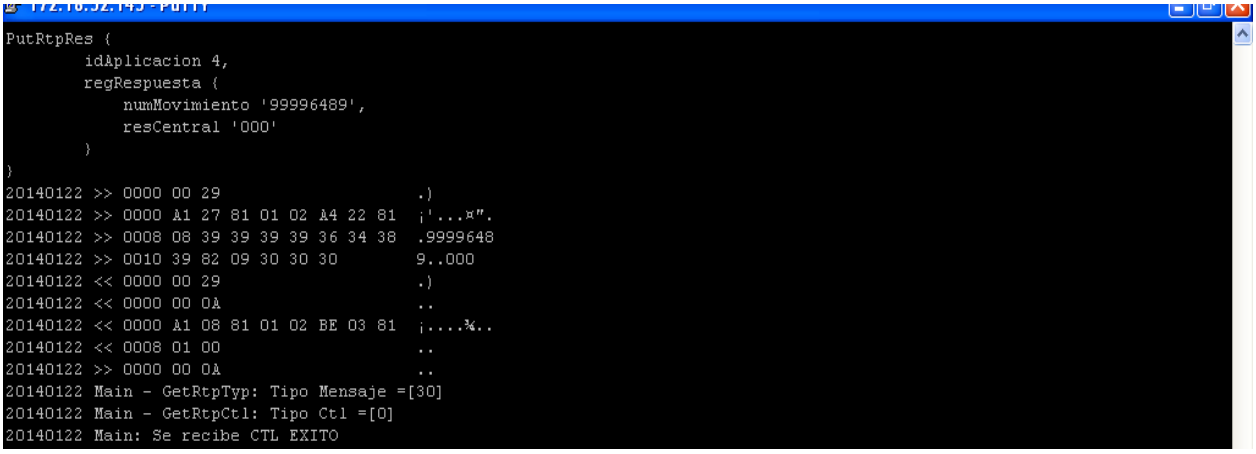


Ilustración 28: Ejemplo cambio teléfono (III)

3.3.2.4. ALTA DE MULTISIM

El alta de líneas multisim llegará por el mismo interfaz que llegan las altas normales de línea con la única diferencia que el tipo de alta que nos llega la recibiremos con valor “MULTISIM”. Con estos valores los campos que recibiremos son:

NOMBRE DEL CAMPO	VALOR
NUM_SECUENCIA	Valor del campo numSecuencia que llega en la trama ASN.1
NUM_TELEFONO	Valor del campo número que llega en la trama ASN.1
NUM_TARJETA	Valor del campo TarjetaSim que llega en la trama ASN.1
TIP_LINEA	Valor del campo TipoLinea que llega en la trama ASN.1
COD_OPERACION	En caso de alta normal valor “ALTA_NORMAL” en caso de línea multisim “ALTA_MULTISIM”
MULTISIM	Valor del campo Lineas_adicionales que llega en la trama ASN.1
TIP_ALTA	En este caso tomará el valor “MULTISIM”

El ASN.1 correspondiente que recibiremos quedará de la siguiente forma:

```
20140206 Hijo: RTP_recv - TMR(CLR) =[120] seg
20140206 << 0000 00 5E                    .^
20140206 << 0000 A1 5C 81 01 01 A3 57 81  ¡\...£W.
20140206 << 0008 08 39 39 39 39 36 34 39  .9999649
20140206 << 0010 30 82 09 36 30 30 30 30  0..60000
20140206 << 0018 30 30 30 30 A3 40 81 13  0000£@..
20140206 << 0020 38 39 30 30 30 30 30 30  89000000
20140206 << 0028 30 30 30 30 30 30 30 30  00000000
20140206 << 0030 30 30 31 82 03 43 82 03  001..C..
20140206 << 0038 4D 55 4C 54 49 53 49 4D  MULTISIM
20140206 << 0038 36 30 30 30 30 30 30 30  60000000
20140206 << 0038 32 24 36 30 30 30 30 30  2$600000
20140206 << 0038 30 30 33                 003
20140206 >> 0000 00 5E                    .^
20140206 pid =[0029d0fa]: GetRtpTyp - Tipo Mensaje =[3]
20140206 pid =[0029d0fa]: Recibe MSG_MOV
GetRtpMov {
        idAplicacion 1,
        regMovimiento {
            numSecuencia '99996490',
            numero '600000000',
            movAlta {
                tarjetaSim '89000000000000001',
                tipoLinea 'C',
                tipoAlta 'MULSISIM',
                lineasAdicionales '600000002$600000003'
            }
        }
}
20140206 Hijo: RTP_recv - TMR(CLR) =[120] seg
```

Ilustración 29: Ejemplo alta de multisim (I)

El siguiente proceso que cogerá el control de la tramitación es el proceso GESTOR. Este proceso se encargará de realizar una serie de chequeos y actualizaciones sobre la BBDD. Los chequeos que se realizarán son:

- Verificar que la línea principal está en la tabla TELEFONOS_ALTA como de alta.
- Extraer la cadena de líneas adicionales a dar de alta y mediante un bucle verificar que todas las líneas se encuentran en la tabla de teléfonos libres (TELEFONOS_LIBRES).

Una vez realizados estos chequeos se procede a realizar los cambios sobre la base de datos. Esta vez solo será necesario realizar un cambio y será actualizar la tabla de TELEFONOS_ALTA indicando en el campo MULTISIMS la cadena de líneas a dar de alta. También eliminaremos de TELEFONOS_LIBRES las líneas que van a ser dadas de alta como multisim. Por último actualizaremos la tabla de movimientos de tal forma que el est_movimien=”ENVIA” para que el control de la tramitación lo coja el proceso encargado de mandar la respuesta al sistema gestor.

El estado final de la base de datos será:

```
SQL> SELECT * FROM MOVIMIENTOS WHERE NUM_SECUENCIA=  99996490;

600000000                      99996490        ALTA_MULTISIM
                       ENVIA       C    06/02/14


Peticion Usuario


               600000002$600000003

SQL>
SQL>
SQL>
SQL>
SQL> select * from TELEFONOS_ALTA where NUM_TELEFONO='600000000';

600000000    8000000000000000000      AE       A    22/10/2013
        600000002$600000003

SQL>
SQL>
SQL>
```

Ilustración 30: Ejemplo alta de multisim (II)

Por último en el módulo “ENVIA” se provisionará la respuesta hacia el sistema gestor indicándole que todo ha ido correctamente.

```
PutRtpRes {
        idAplicacion 1,
        regRespuesta {
            numMovimiento '99996490',
            resCentral '000'
        }
}
20140206 >> 0000 00 29                    .)
20140206 >> 0000 A1 27 81 01 02 A4 22 81  ¡'...¤".
20140206 >> 0008 08 39 39 39 39 36 34 39  .9999649
20140206 >> 0010 30 82 09 30 30 30        0..000
20140206 << 0000 00 29                    .)
20140206 << 0000 00 0A                    ..
20140206 << 0000 A1 08 81 01 02 BE 03 81  ¡....¾..
20140206 << 0008 01 00                    ..
20140206 >> 0000 00 0A                    ..
20140206 Main - GetRtpTyp: Tipo Mensaje =[30]
20140206 Main - GetRtpCtl: Tipo Ctl =[0]
20140206 Main: Se recibe CTL EXITO
20140206
```

Ilustración 31: Ejemplo alta de multisim (III)

Es necesario indicar que para el caso de las bajas de multisim llegarán mediante el interfaz de BAJA llegaría una operación de baja de multisim (COD_OPERACION='BAJA_MULTISIM'). En los chequeos se extraería la cadena y sería necesario verificar que efectivamente la línea indicada a dar de baja se encuentra dentro de las líneas de alta de la línea principal. Una vez verificado esto se eliminarían de la cadena de tal forma que sólo se quedarían las líneas finales.

4. PRUEBAS

En este apartado se van a indicar las distintas pruebas de verificación de software. Se ha dividido el catálogo de pruebas en 3 apartados: pruebas integradas, pruebas unitarias y pruebas de interfaz.

4.1. PRUEBAS INTEGRADAS

Como pruebas integradas nos referimos a aquellas pruebas que se deben realizan desde el inicio de la ejecución (llegada de una petición) hasta que es enviada la respuesta al solicitante.

- CPI001 – ALTA DE LINEA:

Caso de prueba:	*Alta de línea*
Código:	*CPI001*
Descripción:	Llegada a nuestro sistema de una operación de alta de línea de contrato. Se deberá verificar que nos llegan los parámetros num_telefono, num_tarjeta_sim, num_secuencia, tipo_linea.
Requisito funcional:	**RF01**.
Validaciones:	Se debe verificar que el teléfono se inserta en la tabla de teléfonos de alta y se elimina de teléfonos libres, que la tarjeta se elimina de las tarjetas libres y que el movimiento se ha eliminado de la tabla de movimientos y se ha insertado en históricos.
Postcondiciones:	*La línea queda dada de alta en el sistema.*
Observaciones	

- CPI002 - BAJA DE LINEA:

Caso de prueba:	*Baja de línea*
Código:	*CPI002*
Descripción:	Llegada a nuestro sistema de una operación de baja de línea de contrato. Se deberá verificar que nos llegan los parámetros num_telefono, num_tarjeta_sim, num_secuencia, causa_baja.
Requisito funcional:	**RF02**.
Validaciones:	Se debe verificar que el teléfono se elimina en la tabla de teléfonos de alta y se inserta en la de teléfonos libres, que la tarjeta se inserta en la tabla de tarjetas libres y que el movimiento se ha eliminado de la tabla de movimientos y se ha insertado en históricos.
Postcondiciones:	*La línea se da de baja en el sistema.*
Observaciones	

- CPI003 - CAMBIO DE TELEFONO:

Caso de prueba:	*Cambio de teléfono*
Código:	*CPI003*
Descripción:	Llegada a nuestro sistema de una operación de cambio de telefono. Se deberá verificar que nos llegan los parámetros num_telefono_viejo, num_tarjeta_sim, num_telefono_nuevo, num_secuencia.
Requisito funcional:	**RF03**.

Validaciones:	Se debe verificar que el teléfono antiguo se elimina en la tabla de teléfonos de alta y se inserta en la de teléfonos libres, que el teléfono nuevo sustituye al viejo tanto en la tabla de teléfonos de alta como en la tabla de servicios para todos aquellos servicios que tenga activada la linea y que el movimiento se ha eliminado de la tabla de movimientos y se ha insertado en históricos.
Postcondiciones:	*Se realiza la modificación de la linea.*
Observaciones	

- CPI004 – ACTIVACION DE SERVICIO:

Caso de prueba:	*Activación de servicio*
Código:	*CPI004*
Descripción:	Llegada a nuestro sistema de una operación de activación de un servicio. Recibiremos como parámetros de entrada el número de teléfono del usuario, el cod_servicio a dar de alta y la acción (en este caso A:activación).
Requisito funcional:	**RF04**.
Validaciones:	Se debe verificar que se inserta un registro en la tabla de servicios con el servicio que se desea activar y la línea para la que se activa. También se debe verificar que el movimiento se ha eliminado de la tabla de movimientos y se ha insertado en históricos.
Postcondiciones:	*Se realiza la activación del servicio.*
Observaciones	

- CPI005 – DESACTIVACION DEL SERVICIO:

Caso de prueba:	*Desactivación de servicio*
Código:	*CPI004*
Descripción:	Llegada a nuestro sistema de una operación de desactivación de un servicio. Recibiremos como parámetros de entrada el número de teléfono del usuario, el cod_servicio a dar de baja y la acción (en este caso D: desactivación).
Requisito funcional:	**RF04**.
Validaciones:	Se debe verificar que se elimina la marca del servicio de la tabla de servicios. También se debe verificar que el movimiento se ha eliminado de la tabla de movimientos y se ha insertado en históricos.
Postcondiciones:	*Se realiza la desactivación del servicio.*
Observaciones	

- CPI007 – CONSULTA DATOS DE LINEA:

Caso de prueba:	*Consulta de datos de una línea principal*
Código:	*CPI007*
Descripción:	Invocación del tuxedo de consulta de datos de una línea. Recibiremos como parámetros de entrada la línea a consultar. Como parámetros de salida enviaremos el número de tarjeta sim, la fecha de alta y la línea sobre la que se hace la consulta
Requisito funcional:	**RF05**.
Validaciones:	Se debe verificar que se han devuelto correctamente todos los parámetros.
Postcondiciones:	*Se realiza la consulta de la linea.*
Observaciones	

- CPI008 – CONSULTA LINEAS ADICCIONALES:

Caso de prueba:	*Consulta de datos de todas las líneas adicionales de una línea principal*
Código:	*CPI008*
Descripción:	Invocación del tuxedo de consulta de datos de las líneas adicionales. En esta invocación recibiremos como parámetro de entrada la línea principal. Los parámetros de salida serán las distintas líneas adicionales con la fecha de alta de las mismas.
Requisito funcional:	**RF07**.

Validaciones:	Se debe verificar que se han devuelto correctamente todos los parámetros de la consulta.
Postcondiciones:	*Se realiza la consulta de las líneas adicionales de una línea principal.*
Observaciones	

- CPI009 – CONSULTA SERVICIOS:

Caso de prueba:	*Consulta de los servicios para un usuario*
Código:	*CPI009*
Descripción:	Invocación del tuxedo de consulta de servicios activados para una línea. Recibiremos como parámetro de entrada la línea a consultar. Los parámetros de salida serán los distintos servicios que tiene la línea junto con la fecha de alta de cada uno de ellos
Requisito funcional:	**RF08**.
Validaciones:	Se debe verificar que se han devuelto correctamente todos los parámetros de la consulta.
Postcondiciones:	*Se realiza la consulta de los servicios activados de una línea.*
Observaciones	

4.2. PRUEBAS UNITARIAS

En estas pruebas se describen aquellos casos que nos servirán para verificar la correcta gestión de errores de la aplicación. Se deben verificar todos los errores que deberán ser devueltos por la aplicación.

- CPU001 – ALTA LINEA – ERROR TELEFONO YA DADO DE ALTA (CODIGO 1):

Caso de prueba:	ALTA LINEA – ERROR TELEFONO YA DADO DE ALTA (CODIGO 1)
Código:	CPU001
Descripción:	Alta de una línea que ya se encuentra dada de alta en el sistema. Se producirá un error y se retornará un código de error 1.
Requisito funcional:	**RF01**
Validaciones:	Se debe verificar que se produce un fallo ya que el teléfono ya está dado de alta validando que se devuelve el código de error 1.
Postcondiciones:	*Correcta provisión del error a la plataforma.*
Observaciones	

- CPU002 – ALTA LINEA – ERROR TARJETA YA USADA (CODIGO 2):

Caso de prueba:	ALTA LINEA – ERROR TARJETA YA USADA (CODIGO 2)
Código:	CPU002
Descripción:	Alta de una línea cuyo número de tarjeta ya está en uso. Se producirá un error y se retornará un código de error 2.

Requisito funcional:	**RF01**
Validaciones:	Se debe verificar que se produce un fallo ya que el cuyo número de tarjeta ya está en uso, validando que se devuelve el código de error 1.
Postcondiciones:	*Correcta provisión del error a la plataforma.*
Observaciones	

- CPU003 – ALTA ADICIONAL – ERROR LINEA PRINCIPAL NO DADA DE ALTA (CODIGO 1):

Caso de prueba:	ALTA ADICIONAL – ERROR LINEA PRINCIPAL NO DADA DE ALTA (CODIGO 1)
Código:	CPU003
Descripción:	Alta de una línea adicional cuya línea principal no está dada de alta en el sistema. Se producirá un error y se retornará un código de error 1.
Requisito funcional:	**RF01**
Validaciones:	Se debe verificar que se produce un fallo ya que la línea principal a la que va a pertenecer la línea secundaria no está dada de alta, validando que se devuelve el código de error 1.
Postcondiciones:	*Correcta provisión del error a la plataforma.*
Observaciones	

- CPU004 – ALTA ADICIONAL – ERROR LINEA ADICIONAL YA DADA DE ALTA (CODIGO 2):

Caso de prueba:	ALTA ADICIONAL – ERROR LINEA ADICIONAL YA DADA DE ALTA (CODIGO 1)
Código:	CPU004
Descripción:	Alta de una línea adicional que ya está dada de alta en el sistema. Se producirá un error y se retornará un código de error 2.
Requisito funcional:	**RF01**
Validaciones:	Se debe verificar que se produce un fallo ya que la línea adicional a dar de alta ya lo está, validando que se devuelve el código de error 2.
Postcondiciones:	*Correcta provisión del error a la plataforma.*
Observaciones	

- CPU005 – BAJA DE LINEA – ERROR LINEA YA DADA DE BAJA (CODIGO 1):

Caso de prueba:	BAJA DE LINEA – ERROR LINEA YA DADA DE BAJA (CODIGO 1)
Código:	CPU005
Descripción:	Baja de línea de un teléfono que ya se encuentra dado de baja. El sistema debe retornar un código de error 1.
Requisito funcional:	**RF02**
Validaciones:	Se debe verificar que se produce un fallo ya que la línea a dar de baja no está dada de alta en el sistema y que se envía un código de error 1.
Postcondiciones:	*Correcta provisión del error a la plataforma.*

- CPU006 – CAMBIO DE LINEA – ERROR TELEFONO A CAMBIAR NO DADO DE ALTA (CODIGO DE ERROR 1):

Caso de prueba:	CAMBIO DE LINEA – ERROR TELEFONO A CAMBIAR NO DADO DE ALTA (CODIGO DE ERROR 1)
Código:	CPU006
Descripción:	Cambio de línea de un teléfono que no se encuentra dado de alta en el sistema. Se retornará el código de error 1.
Requisito funcional:	**RF02**
Validaciones:	Se debe verificar que se produce un fallo ya que la línea que se desea modificar no está dada de alta en el sistema y que se responde con un código de error 1.
Postcondiciones:	*Correcta provisión del error a la plataforma.*

- CPU007 – CAMBIO DE LINEA – ERROR NUEVA LINEA YA ESTA DADA DE ALTA (CODIGO DE ERROR 2):

Caso de prueba:	CAMBIO DE LINEA – ERROR NUEVA LINEA YA ESTA DADA DE ALTA (CODIGO DE ERROR 2)
Código:	CPU007
Descripción:	Cambio de línea a una línea que ya se encuentra dada de alta en el sistema. Se retornará el código de error 2.
Requisito funcional:	**RFXX**
Validaciones:	Se debe verificar que se comprueba que la nueva línea adicional ya se encuentra dada de alta por lo que no se puede dar de alta y se validará que se retorna el código de error 2.
Postcondiciones:	*Correcta provisión del error a la plataforma.*

- CPU008 – ACTIVACION DE SERVICIO – ERROR LINEA CUYO SERVICIO SE QUIERE ACTIVAR NO DADA DE ALTA EN EL SISTEMA (CODIGO DE ERROR 2):

Caso de prueba:	ACTIVACION DE SERVICIO – ERROR LINEA CUYO SERVICIO SE QUIERE ACTIVAR NO DADA DE ALTA EN EL SISTEMA (CODIGO DE ERROR 2)
Código:	CPUXX
Descripción:	Activación / desactivación de un servicio a una línea dada de alta. Se comprobará que la línea no está dada de alta en el sistema y se retornará un código de error 1.
Requisito funcional:	**RFXX**
Validaciones:	Se debe verificar que la línea a la cual se le quiere gestionar un servicio no está dada de alta en el sistema, validando que se devuelve un código de error 1.
Postcondiciones:	*Correcta provisión del error a la plataforma.*
Observaciones	

- CPU009 – CONSULTA DATOS LINEA – ERROR LINEA NO DADA DE ALTA EN EL SISTEMA (CODIGO DE ERROR 1):

Caso de prueba:	CONSULTA DATOS LINEA – ERROR LINEA NO DADA DE ALTA EN EL SISTEMA (CODIGO DE ERROR 1)
Código:	CPUXX
Descripción:	Al realizar la consulta sobre una línea se verá que dicha línea no existe y se retornará un código de error 1.
Requisito funcional:	**RFXX**
Validaciones:	Se debe verificar que la línea que se quiere consultar no existe y se validará que se retorna un código de error 1.
Postcondiciones:	*Correcta provisión del error a la plataforma.*

- CPU010 – CONSULTA LINEAS ADICIONALES – ERROR LINEA PRINCIPAL NO EXISTENTE EN EL SISTEMA (CODIGO DE ERROR 1):

Caso de prueba:	CONSULTA DATOS LINEA – ERROR LINEA NO DADA DE ALTA EN EL SISTEMA (CODIGO DE ERROR 1)
Código:	CPUXX
Descripción:	Realizar una consulta sobre las líneas adicionales de una línea principal. Se comprobará que la línea principal no está dada de alta y se retornará un código de error 1.
Requisito funcional:	**RFXX**
Validaciones:	Se debe verificar que la línea principal no existe en el sistema y se validará que se retorna un código de error 1.
Postcondiciones:	*Correcta provisión del error a la plataforma.*
Observaciones	

- CPU011 – CONSULTA SERVICIOS – ERROR LINEA PRINCIPAL NO EXISTENTE EN EL SISTEMA (CODIGO DE ERROR 1):

Caso de prueba:	CONSULTA SERVICIOS – ERROR LINEA PRINCIPAL NO EXISTENTE EN EL SISTEMA (CODIGO DE ERROR 1)
Código:	CPUXX
Descripción:	Realizar una consulta sobre los servicios activados de una línea. Se comprobará que la línea a consultar no existe en el sistema y se retornará un código de error 1.
Requisito funcional:	**RFXX**
Validaciones:	Se debe verificar que la línea no existe en el sistema y se validará que se retorna un código de error 1.

Postcondiciones:	*Correcta provisión del error a la plataforma.*
Observaciones	

4.3. PRUEBAS DE INTERFAZ

En este apartado se describirán aquellas pruebas que nos valdrán para verificar la conexión del sistema usuario con el nuestro. Para esta colección de pruebas será necesario validar tanto la conectividad del sistema con el nuevo servidor tuxedo, así como la conectividad del sistema remoto con los distintos listeners (procesos recibe) de nuestro lote de procesos BATCH.

- CPI001 – CONECTIVIDAD SERVIDOR TUXEDO:

Caso de prueba:	VERIFICACION CONECTIVIDAD SISTEMA REMOTO – SERVIDOR TUXEDO
Código:	CPI001
Descripción:	Verificar que es posible la conexión del sistema remoto con nuestro servidor tuxedo. Para la realización de esta prueba el sistema remoto lanzará un telnet hasta la máquina donde se encuentre instalada la aplicación al puerto sobre que esté escuchando nuestro servidor. Ej: telnet 111.111.111.111 80880
Requisito funcional:	**RFXX**
Validaciones:	Verificar que una vez realizado el telnet la conexión se queda abierta.
Postcondiciones:	*Conexión establecida.*
Observaciones	

- CPI002 – CONECTIVIDAD PROCESO BATCH:

Caso de prueba:	PRUEBA CONECTIVIDAD PROCESO BATCH
Código:	CPI002
Descripción:	Se realizará el telnet sobre la máquina y el puerto en que estará escuchando nuestro proceso recibe. Ej: telnet 111.111.111.111 80880
Requisito funcional:	**RFXX**
Validaciones:	Verificar que se establece la conexión.
Postcondiciones:	*Conexión establecida.*
Observaciones	

5. CONCLUSIONES Y TRABAJOS FUTUROS

5.1. CONCLUSIONES

Una vez finalizado el estudio, podemos decir que en este proyecto se han ejecutado y cumplido todos los objetivos marcados en un principio, así como los distintos requisitos funcionales y de interfaz de los que partíamos. Nos hemos basado casi completamente en el diseño marcado en un principio sin desviarnos de la idea principal y respetando las características del diseño.

Se ha desarrollado el proyecto del sistema de provisión en red para líneas móviles de principio a fin, identificando el mercado afectado, desarrollando los distintos requisitos del sistema, realizando una correcta gestión de proyecto, un buen análisis, una programación respetando los estándares y por último una batería de pruebas amplia y robusta que nos ha permitido verificar la integridad del sistema.

Hemos conseguido el desarrollo de un sistema estándar, fácil de adaptar y muy económico lo que permite ser un software interesante desde el punto de vista del mercado actual.

En el ámbito personal del proyecto puedo indicar que este desarrollo me ha servido para mejorar mis destrezas en los lenguajes desarrollados. Así mismo debo destacar el gran aprendizaje obtenido al llevar la ejecución de un proyecto desde su inicio con el requisito maestro, pasando por el estudio de mercado, diseño e implementación del sistema y pruebas lo que es un gran aprendizaje de cara a la vida laboral.

5.2. TRABAJOS FUTUROS

También queremos dejar un espacio para indicar una serie de mejores que pueden ser abordadas en un futuro de cara a potenciar el sistema y aumentar su funcionalidad.

- Uno de las posibles mejoras a abordar puede ser la incorporación de líneas corporativas. Actualmente este tipo de líneas suponen un amplio mercado y el volumen de demanda es elevado. Esta mejora debería acometerse una vez este sistema estuviera consolidado ya que supone un fuerte riesgo.
- Dentro de este diseño no se incluyen todo aquella funcionalidad relativa a la gestión de tarifas, tanto contratación como bajas y consultas. En un principio este apartado deberá ser desarrollado de forma paralela a nuestro desarrollo, pero debe plantearse la integración de esta funcionalidad dentro de nuestro sistema en un futuro.
- Actualmente con este proyecto se abarca sólo el mercado de líneas móviles (tanto prepago como residenciales) pero función del éxito del mismo se podría expandir al mercado de líneas fijas. El proceso para integrar estas funcionalidades dentro de nuestro sistema no sería excesivamente complejo.
- También se debe indicar que este sistema no se va a encargar de la tarificación de líneas ni del consumo. En un futuro consideramos oportuno evaluar la posibilidad de incluir un sistema de tarificación dentro del sistema de cara a centralizar todas las funcionalidades de la aplicación.

6. ANEXOS

6.1. PLANIFICACIÓN DEL PROYECTO

6.1.1.COSTE DEL PROYECTO

Para poder realizar una correcta gestión del proyecto debemos realizar una estimación, tanto de costes como de tiempo. Para llevar a cabo esta estimación vamos a utilizar el método COCOMO. Este método está basado en una serie de algoritmos matemáticos que en función de las líneas de código implementadas nos sirvan para calcular tanto la amplitud del proyecto, como la duración del mismo y el esfuerzo de todo el equipo de trabajo.

En primer lugar debemos indicar que hemos elegido un ciclo de vida en cascada:

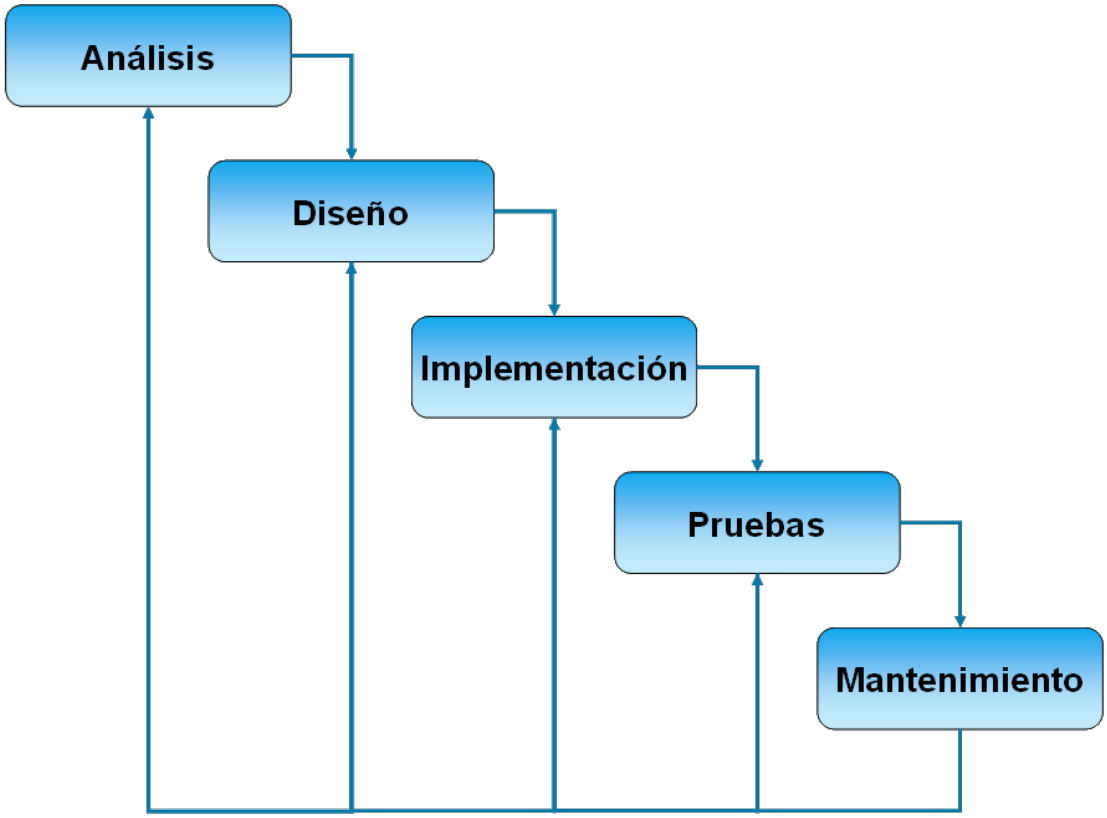


Ilustración 32: Diagrama ciclo de vida en cascada

Este método se encarga de dividir el proyecto en 5 partes diferenciadas:

- Etapa de análisis: identificación de los distintos requisitos exigibles para el correcto funcionamiento del proyecto.
- Etapa de diseño: planteamiento de la solución que se va a llevar a cabo para el cumplimiento de cada uno de los requisitos especificados.
- Etapa de implementación: creación del código fuente en función del diseño creado en la anterior etapa.
- Etapa de pruebas: diseño y ejecución de un amplio abanico de pruebas (tanto integradas como unitarias) de cara a realizar una validación del software realizado.
- Etapa de mantenimiento: finalización del proyecto. Resolución de posibles incidencias del sistema, correctivos a implementar y posibles evolutivos de cara a la mejora de la aplicación.

6.1.2. ESTIMACION

Vamos a basarnos en un tipo de desarrollo orgánico, ya que no vamos a realizar ninguna innovación y nos vamos a basar en funcionalidades ya existentes, la documentación está muy especificada y las pruebas se encuentran bien definidas. Vamos a obtener los valores de las variables a y b a partir de la siguiente tabla:

MODO	a	b	c	d
Orgánico	2.40	1.05	2.50	0.38
Semilibre	3	1.12	2.50	0.35
Rígido	3.60	1.20	2.50	0.32

Ilustración 33: Tabla calibrado tipos de desarrollo

Para identificar el número de líneas de código implementadas, hemos identificado dos partes: una para la parte Batch y otro para la parte Tuxedo (Online).

- Parte Batch: hemos identificado 10 módulos creados con un cálculo aproximado de 500 líneas por módulo desarrolladas tanto en C como en pro*C.
- Parte Tuxedo: existen dos nuevos servicios tuxedo, estimamos un esfuerzo totas de 500 líneas de código para cada servicio (1000 líneas en total) desarrolladas en C++.

Con este cálculo obtenemos un total de 6000 líneas de código para la aplicación completa.

6.1.3. ESFUERZO BRUTO

Para realizar el cálculo del esfuerzo bruto nos vamos a fijar en la siguiente fórmula:

$$PMb = a \cdot tamaño^{b}$$

En la que b=1,05, a=2,40 y tamaño=6 Klineas, por lo tanto:

$$PMb = 2{,}40 * 6^{1{,}05}$$

Por lo que obtenemos un total de **15,75 personas por mes**.

6.1.4. ESFUERZO AFINADO

Como hemos optado por un desarrollo orgánico, debemos aplicar el mayor valor de influencia para nuestro desarrollo, en este caso el factor RELY, por lo que aplicaremos un factor de 1,29 al valor obtenido anteriormente para calcular el valor del esfuerzo afinado.

FACTORES CORRECTORES	NOMBRE	COEFICIENTE	INFLUENCIA
Aptitud del equipo	ECAP	2.06	14
Complejidad del producto	CLPX	1.45	2
Experiencia en áreas de aplicación	AEXP	1.19	8
Experiencia en el lenguaje	LEXP	1.07	9
Experiencia en la máquina virtual	VEXP	1.11	6
Inestabilidad de la máquina virtual	VIRT	1.16	7
Inestabilidad de las especificaciones	REQU	1.23	13
Requisitos de fiabilidad	**RELY**	**1.29**	**1**
Restricciones de duración durante el proyecto	SCED	1.08	12
Restricciones de tamaño de memoria	STOR	1.19	4
Restricciones de tiempo de ejecución	TIME	1.22	10
Técnicas de desarrollo evolucionadas	MODP	1.31	3
Utilización de herramientas software	TOOL	1.22	11
Volumen de datos manipulado	DATA	1.08	5

Ilustración 34: Tabla características esfuerzo afinado

Aplicando la fórmula que mostramos a continuación podremos realizar el cálculo del esfuerzo ajustado:

$$PMa = Coeficiente\ corrector * PMb$$

Por lo que, si realizamos las sustituciones oportunas, obtenemos:

$$PMa = 1{,}29 * 15{,}75$$

Obtenemos un resultado de **20,32 personas por mes**

6.1.5. CÁLCULO DEL TIEMPO PLANIFICADO

Ahora procederemos a calcular el tiempo estimado de puesta en marcha del proyecto.

Para ello usaremos la fórmula siguiente:

$$Tiempo = C * PMa^{D}$$

Donde C = 2,50 y D = 0,38, después de realizar la sustitución relativa a la tabla de tipo de desarrollo, que para nuestro caso sería un tipo orgánico.

El valor resultante será = **7,85 meses de duración del proyecto.**

6.1.6. ESTIMACIÓN COSTE DEL PROYECTO

Por último debemos calcular el coste total del proyecto. Para ello en primer lugar deberemos calcular el número de recursos (personas) que serán necesarios para llevar a cabo el desarrollo. Para realizar el cálculo del número de personas debemos aplicar la siguiente ecuación:

$$PM = Personas * Tiempo$$

Por lo que: $\text{Personas} = \text{PM}/\text{Tiempo}$, sustituyendo obtenemos un resultado 2,63 => **3 personas.**

La estimación la vamos a realizar contando con un 50 % de dedicación de los recursos. Estimando que cada mes se realizan 180 horas de trabajo, equivale a 90 horas mensuales de dedicación por parte de cada recurso. Actualmente cada jornada de trabajo estará valorada en 300 €/UTPS por lo que para calcular el precio por hora de cada recurso realizamos el siguiente cálculo:

$$PPH = PUTPS/HDiarias$$

Partiendo de la base de que cada jornada consta de 8 horas obtenemos un valor total de **37.5 € /Hora de trabajo.**

Para evaluar el coste total del proyecto debemos calcular el coste mensual y realizar la multiplicación por el número de meses del proyecto. El coste mensual se calculará de la siguiente manera:

$$C = PPH * \frac{Horas}{mes} * nºrecursos$$

Como tenemos un precio por hora de 37,5, una dedicación del 50% sobre 180 jornadas (90 jornadas) y un total de 3 recursos obtendremos un valor total de 10125 € /mes. Como hemos estimado una duración total de 7,85 meses, podemos estimar el Coste total del proyecto de la siguiente forma:

$$CTotal = C * Tiempo$$

Por lo que si sustituimos obtenemos que el coste total aproximado del proyecto será 10125€/mes * 7,85 meses = 79481,25 €

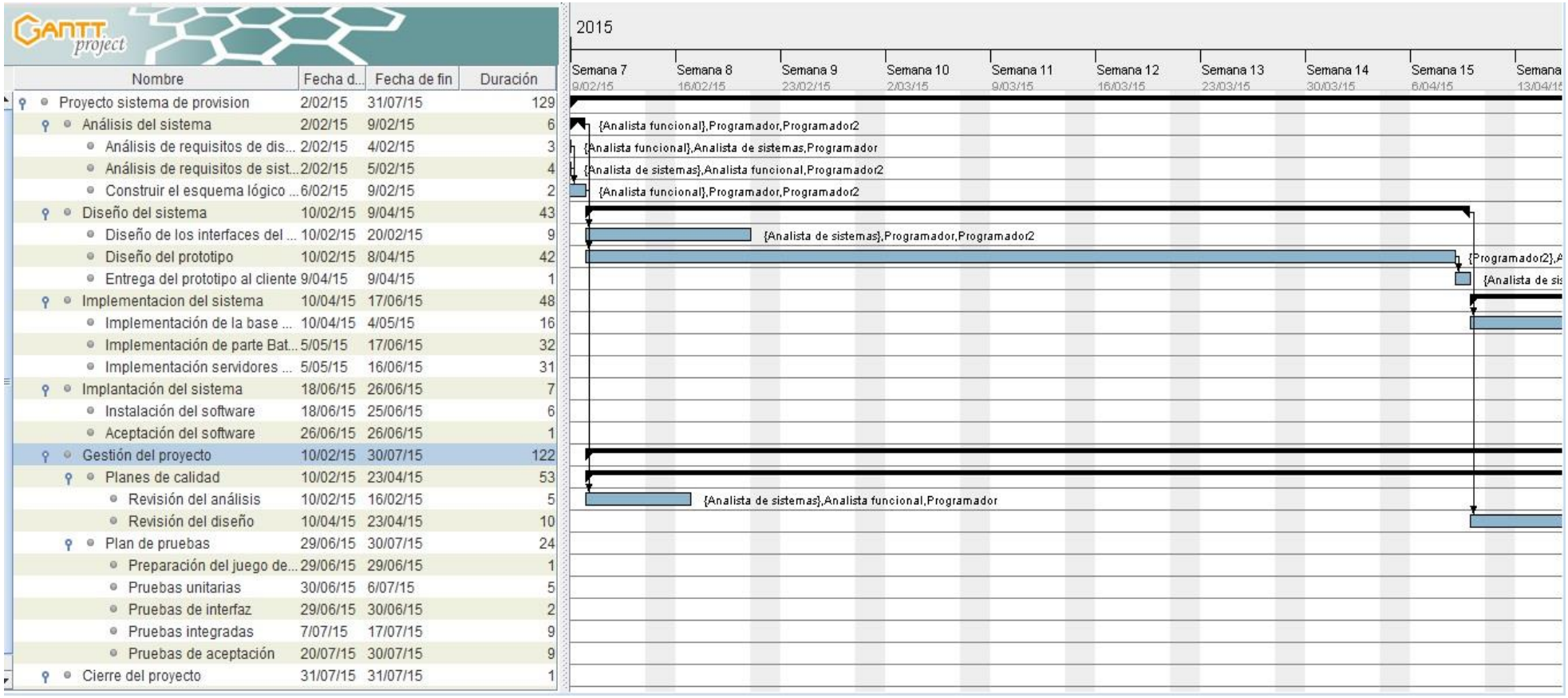
COSTE TOTAL = 79481,25 €

Ahora mostraremos el desglose de costes y tiempo en función de las distintas etapas del desarrollo. Las 5 etapas definidas son: la etapa de análisis, la etapa de diseño, la de programación, la de pruebas y la de puesta en producción (implantación). Hemos asignado un porcentaje a cada una de estas etapas por lo que el desglose quedará de la siguiente manera:

FASE	COSTE (en %)	Duración (en %)	Coste	Duración
Análisis	10 %	5 %	7948,12 €	35,32 Horas
Diseño	35 %	30 %	27818,44 €	211,95 Horas
Programación	40 %	45 %	31792,50 €	317,93 Horas
Pruebas	10 %	15 %	7948,12 €	105,98 Horas
Implantación	5 %	5 %	3974,06 €	35,32 Horas

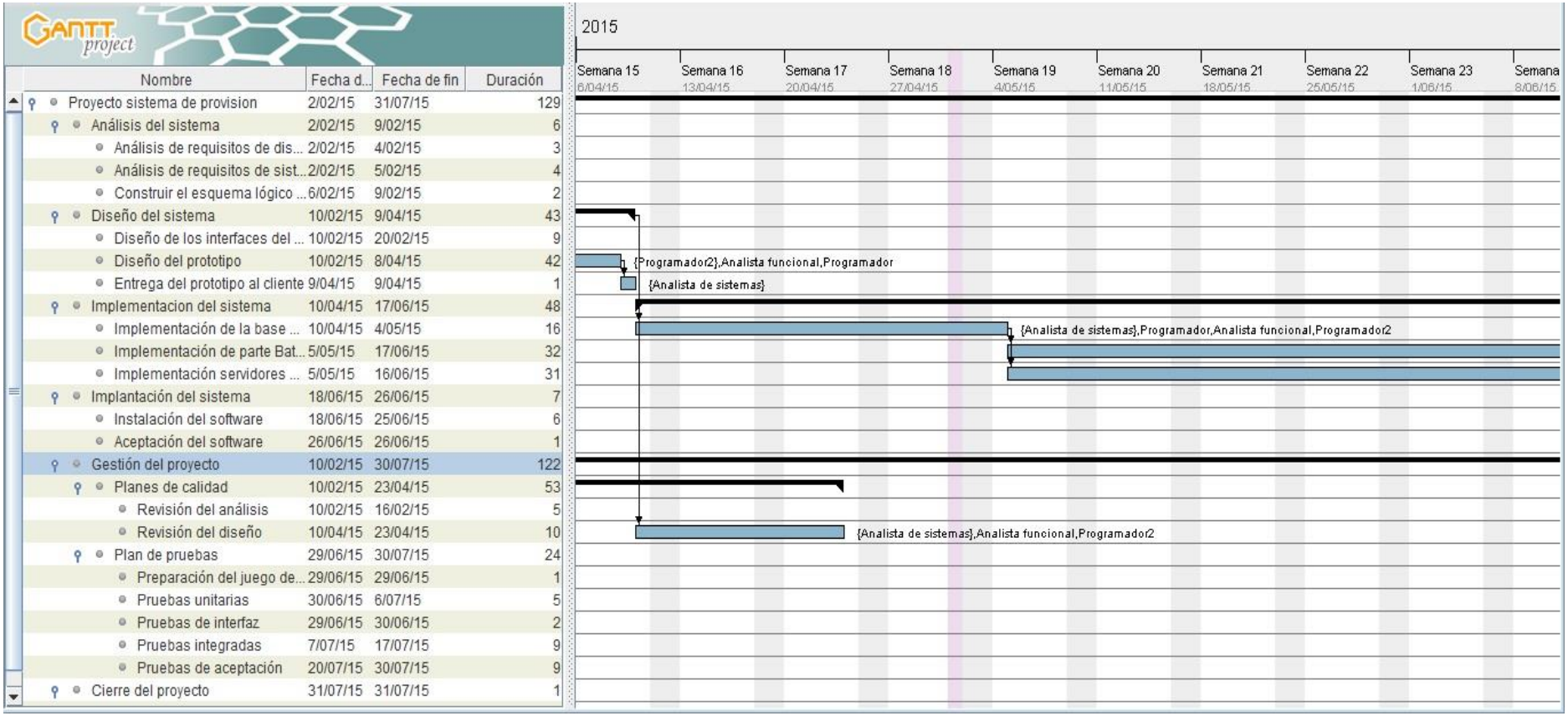
Ilustración 35: Coste fases del desarrollo

6.1.7. GRÁFICO GANTT



Nombre	Fecha d...	Fecha de fin	Duración
Proyecto sistema de provision	2/02/15	31/07/15	129
Análisis del sistema	2/02/15	9/02/15	6
Análisis de requisitos de dis...	2/02/15	4/02/15	3
Análisis de requisitos de sist...	2/02/15	5/02/15	4
Construir el esquema lógico ...	6/02/15	9/02/15	2
Diseño del sistema	10/02/15	9/04/15	43
Diseño de los interfaces del ...	10/02/15	20/02/15	9
Diseño del prototipo	10/02/15	8/04/15	42
Entrega del prototipo al cliente	9/04/15	9/04/15	1
Implementacion del sistema	10/04/15	17/06/15	48
Implementación de la base ...	10/04/15	4/05/15	16
Implementación de parte Bat...	5/05/15	17/06/15	32
Implementación servidores ...	5/05/15	16/06/15	31
Implantación del sistema	18/06/15	26/06/15	7
Instalación del software	18/06/15	25/06/15	6
Aceptación del software	26/06/15	26/06/15	1
Gestión del proyecto	10/02/15	30/07/15	122
Planes de calidad	10/02/15	23/04/15	53
Revisión del análisis	10/02/15	16/02/15	5
Revisión del diseño	10/04/15	23/04/15	10
Plan de pruebas	29/06/15	30/07/15	24
Preparación del juego de...	29/06/15	29/06/15	1
Pruebas unitarias	30/06/15	6/07/15	5
Pruebas de interfaz	29/06/15	30/06/15	2
Pruebas integradas	7/07/15	17/07/15	9
Pruebas de aceptación	20/07/15	30/07/15	9
Cierre del proyecto	31/07/15	31/07/15	1

Ilustración 36: Planificación del proyecto (I)



Nombre	Fecha d...	Fecha de fin	Duración
Proyecto sistema de provision	2/02/15	31/07/15	129
Análisis del sistema	2/02/15	9/02/15	6
Análisis de requisitos de dis...	2/02/15	4/02/15	3
Análisis de requisitos de sist...	2/02/15	5/02/15	4
Construir el esquema lógico ...	6/02/15	9/02/15	2
Diseño del sistema	10/02/15	9/04/15	43
Diseño de los interfaces del ...	10/02/15	20/02/15	9
Diseño del prototipo	10/02/15	8/04/15	42
Entrega del prototipo al cliente	9/04/15	9/04/15	1
Implementacion del sistema	10/04/15	17/06/15	48
Implementación de la base ...	10/04/15	4/05/15	16
Implementación de parte Bat...	5/05/15	17/06/15	32
Implementación servidores ...	5/05/15	16/06/15	31
Implantación del sistema	18/06/15	26/06/15	7
Instalación del software	18/06/15	25/06/15	6
Aceptación del software	26/06/15	26/06/15	1
Gestión del proyecto	10/02/15	30/07/15	122
Planes de calidad	10/02/15	23/04/15	53
Revisión del análisis	10/02/15	16/02/15	5
Revisión del diseño	10/04/15	23/04/15	10
Plan de pruebas	29/06/15	30/07/15	24
Preparación del juego de...	29/06/15	29/06/15	1
Pruebas unitarias	30/06/15	6/07/15	5
Pruebas de interfaz	29/06/15	30/06/15	2
Pruebas integradas	7/07/15	17/07/15	9
Pruebas de aceptación	20/07/15	30/07/15	9
Cierre del proyecto	31/07/15	31/07/15	1

Ilustración 37: Planificación del proyecto (II)

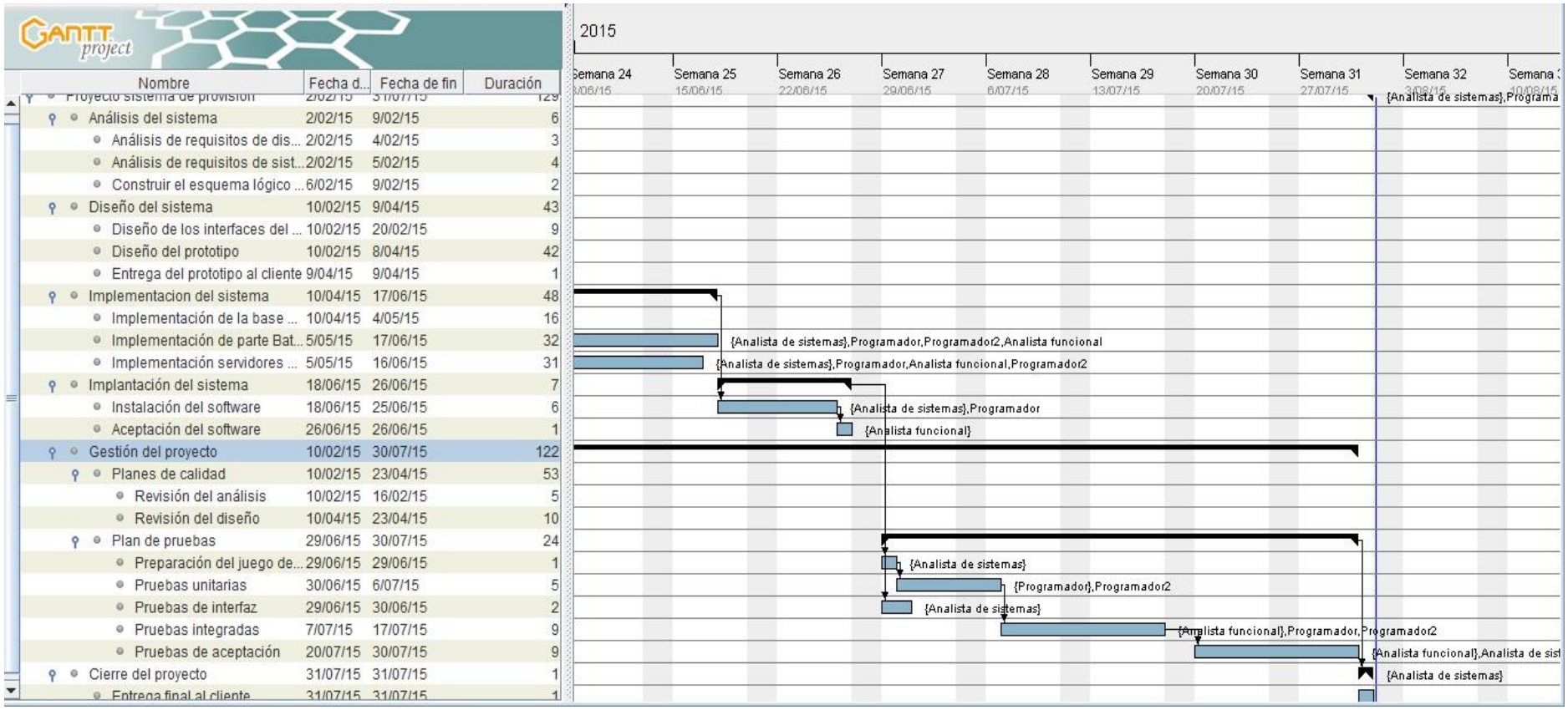


Ilustración 38: Planificación del proyecto (II

6.2. CODIGO DE ERRORES A DEVOLVER AL SISTEMA GESTOR

Se han establecido una serie de códigos de error que serán devueltos al sistema gestor. Los códigos de error diseñados están abiertos a modificaciones y son:

OPERACIÓN	CODIGO DE ERROR	DESCRIPCION ERROR
ALTA DE LINEA	001	Línea no existe en la tabla TELEFONOS_LIBRES
ALTA DE LINEA	002	Tarjeta no existente en la tabla TARJETAS_LIBRES
BAJA DE LINEA	001	Línea no existente en la tabla TELEFONOS_ALTA
ALTA DE SERVICIOS	001	Línea no existente en la tabla TELEFONOS_ALTA
ALTA DE SERVICIOS	002	Servicio ya dado de alta para la línea en la tabla SERVICIOS
BAJA DE SERVICIOS	001	Línea no existente en la tabla TELEFONOS_ALTA
BAJA DE SERVICIOS	003	Servicio ya dado de Baja en la tabla SERVICIOS
CAMBIO DE TELEFONO	001	Línea antigua no existente en TELEFONOS_ALTA
CAMBIO DE TELEFONO	002	Línea nueva no existente en la tabla TELEFONOS_LIBRES
ALTA DE MULTISIM	001	Línea principal no existente en TELEFONOS_ALTA
ALTA DE MULTISIM	002	Líneas adicionales no se encuentran en la tabla TELEFONOS_ALTA
BAJA DE MULTISIM	001	Línea principal no presente en la tabla TELEFONOS_ALTA
BAJA DE MULTISIM	003	Líneas adicionales ya dadas de baja en TELEFONOS_ALTA

Ilustración 39: Tabla listado de errores

6.3. FUNCIONES DE INSERCION, ACTUALIZACION, BUSQUEDA Y BORRADO DE LA BASE DE DATOS

Para realizar los accesos necesarios a BBDD en la aplicación se ha optado por el diseño de una serie de funciones pro *c. Estas funciones han sido diseñadas en lenguaje C++ y permitirán a los distintos procesos BATCH realizar las operaciones necesarias sobre la base de datos (inserción de nuevos registros, actualizaciones de registros, búsqueda de registros y borrado de los mismos).

Para indicar como se han creado estas librerías vamos a mostrar el funcionamiento de una de ellas. El patrón seguido por las demás es muy similar modificando los campos de consulta, claves, etc.

Función de consulta:

```
Funcion int Select_MOVIMIENTO ( movimiento *pt )
{
    int st ;
    ( ( st = Open_MOVIMIENTO ( pt ) ) < 0 ||
      ( st = Read_MOVIMIENTO ( pt ), Close_MOVIMIENTO ( ), st ) < 0 )
;
    return st ;
}
```

Esta función se encarga de realizar 3 operaciones:

- Apertura del cursor que va a buscar el registro en BBDD:

```
int Open_MOVIMIENTO ( movimiento *pt )
    EXEC SQL
        OPEN C_MOVIMIENTO ;
    oraperror ( "Open_MOVIMIENTO" ) ;
    return sqlca.sqlcode ;
```

- Lectura del registro de base de datos. Esta operación se realiza en dos pasos:

  - Ejecución del cursor de búsqueda que va a obtener el registro de base de datos:

```
int Decl_MOVIMIENTO ( void )
{
    EXEC SQL
        DECLARE C_MOVIMIENTO CURSOR FOR
        SELECT
            ROWID,
                  sNUM_TELEFONO,
                  sNUM_TELNUEVO,
                  sNUM_SECUENCIA,
                  sCOD_OPERACION,
                  sNUM_TARJETA,
                  sEST_MOVIMIEN,
                  sTIP_LINEA,
                  TO_CHAR(sFEC_MOVIMIEN,'YYYYMMDDHH24MISS'),
                  sCOD_SERVICIO,
                  sCOD_ERROR,
                  sCAUSA_BAJA,
                  sMULTISIM,
                  cTIP_ACCION_SERVICIO
        FROM  MOVIMIENTOS
        WHERE NUM_SECUENCIA = :sNUM_SECUENCIA
          AND ROWNUM = 1 ;
```

```
    oraperror ( "Decl_MOVIMIENTO" ) ;
    return sqlca.sqlcode ;
```

- Posteriormente recuperamos el registro del cursor mediante el FETCH y guardamos la información en la estructura definida:

```
int Read_MOVIMIENTO ( movimiento *pt )
{
    sMOV_ROWID     = pt->sMOV_ROWID ;
      sNUM_TELEFONO  = pt->sNUM_TELEFONO ;
      sNUM_TELNUEVO  = pt->sNUM_TELNUEVO ;
      sNUM_SECUENCIA = pt->sNUM_SECUENCIA ;
      sCOD_OPERACION = pt->sCOD_OPERACION ;
      sNUM_TARJETA   = pt->sNUM_TARJETA ;
      sEST_MOVIMIEN  = pt->sEST_MOVIMIEN ;
      sTIP_LINEA     = pt->sTIP_LINEA ;
      sFEC_MOVIMIEN  = pt->sFEC_MOVIMIEN ;
      sCOD_SERVICIO  = pt->sCOD_SERVICIO ;
      sCOD_ERROR     = pt->sCOD_ERROR ;
      sCAUSA_BAJA    = pt->sCAUSA_BAJA ;
      sMULTISIM      = pt->sMULTISIM ;
      cTIP_ACCION_SERVICIO = pt->cTIP_ACCION_SERVICIO ;

    EXEC SQL
        FETCH C_MOVIMIENTO
        INTO
            :sMOV_ROWID,
            :sEST_TELEFONO,
                  :sNUM_TELNUEVO,
            :sNUM_SECUENCIA,
                  :sCOD_OPERACION,
            :sNUM_TARJETA,
            :sEST_MOVIMIEN,
                  :sTIP_LINEA,
                  :sFEC_MOVIMIEN,
                  :sCOD_SERVICIO,
                  :sCOD_ERROR,
                  :sCAUSA_BAJA,
                  :sMULTISIM,
                  :cTIP_ACCION_SERVICIO
```

```
    oraperror ( "Read_MOVIMIENTO" ) ;
    if ( sqlca.sqlcode )
        return sqlca.sqlcode ;
    MostrarLinea(  "Read_MOVIMIENTO", pt ) ;
    return sqlca.sqlcode ;
}
```

- Por último tenemos la función de se encarga de cerrar el cursor:

```
int Close_MOVIMIENTO ( void )
{
#ifndef _NO_CLOSE_
    EXEC SQL
        CLOSE C_MOVIMIENTO ;
    oraperror ( "Close_MOVIMIENTO" ) ;
    return sqlca.sqlcode ;
#else
    return 0 ;
#endif
}
```

Función de inserción en BBDD:

Para la inserción del registro en base de datos hemos creado una función simple que crea un insert embebido dentro del código c. La función creada es la siguiente:

```
int Insert_MOVIMIENTO ( movimiento *pt )
{
      sNUM_TELEFONO  = pt->sNUM_TELEFONO ;
      sNUM_TELNUEVO  = pt->sNUM_TELNUEVO ;
      sNUM_SECUENCIA = pt->sNUM_SECUENCIA ;
      sCOD_OPERACION = pt->sCOD_OPERACION ;
      sNUM_TARJETA   = pt->sNUM_TARJETA ;
      sEST_MOVIMIEN  = pt->sEST_MOVIMIEN ;
      sTIP_LINEA     = pt->sTIP_LINEA ;
      sFEC_MOVIMIEN  = pt->sFEC_MOVIMIEN ;
      sCOD_SERVICIO  = pt->sCOD_SERVICIO ;
      sCOD_ERROR     = pt->sCOD_ERROR ;
      sCAUSA_BAJA    = pt->sCAUSA_BAJA ;
      sMULTISIM      = pt->sMULTISIM ;
```

```
      cTIP_ACCION_SERVICIO = pt->cTIP_ACCION_SERVICIO ;

    MostrarLinea (  "Insert_MOVIMIENTO", pt ) ;
    EXEC SQL
        INSERT INTO MOVIMIENTOS (
            NUM_TELEFONO,
                 NUM_TELNUEVO,
            NUM_SECUENCIA,
                 COD_OPERACION,
            NUM_TARJETA,
            EST_MOVIMIEN ,
            TIP_LINEA ,
                 FEC_MOVIMIEN ,
                 COD_SERVICIO ,
                 COD_ERROR ,
                 CAUSA_BAJA ,
                 MULTISIM ,
                 TIP_ACCION_SERVICIO
        )
        VALUES (
           :sNUM_TELEFONO,
               :sNUM_TELNUEVO,
           :sNUM_SECUENCIA,
               :sCOD_OPERACION,
           :sNUM_TARJETA,
           :sEST_MOVIMIEN,
           :sTIP_LINEA,
               :sFEC_MOVIMIEN,
               :sCOD_SERVICIO,
               :sCOD_ERROR,
               :sCAUSA_BAJA,
               :sMULTISIM,
               :cTIP_ACCION_SERVICIO,
        ) ;
    oraperror ( "Insert_MOVIMIENTO" ) ;
    return sqlca.sqlcode ;
}
```

Función de borrado en BBDD:

Para la función de borrado de registros se ha creado una función que se encargar de la búsqueda del registro a eliminar y proceder a eliminarlo. La función tiene la siguiente implementación:

```
int Delete_MOVIMIENTO ( movimiento *pt )
{
    sNUM_SECUENCIA = pt->sNUM_SECUENCIA ;

    EXEC SQL
        DELETE MOVIMIENTOS
        WHERE NUM_SECUENCIA  = :sNUM_SECUENCIA ;

    oraperror ( "Delete_MOVIMIENTO" ) ;
    return sqlca.sqlcode ;
}
```

Función de actualización en BBDD:

Las funciones diseñadas para la funcionalidad de actualización sobre base de datos sí que tienen bastantes cambios entre las distintas tablas. Cabe indicar que en varios casos ha sido necesaria la creación de más de una función de actualización ya que los variables a actualizar pueden variar en función de la parte de la tramitación en que estés ubicado. Como ejemplo mostramos la implementación de la actualización de un movimiento. Esta función se ejecuta siempre que un movimiento ha finalizado su tramitación en un estado y debe pasar el control a otro estado (RECIBE -> GESTOR-> ENVIA):

```
int Update_MOVIMIENTO ( movimiento *pt )
{
    sMOV_ROWID     = pt->sMOV_ROWID ;
      sEST_MOVIMIEN  = pt->sEST_MOVIMIEN ;
    sCOD_ERROR     = pt->sCOD_ERROR ;

    MostrarMovimiento (  "Update_MOVIMIENTO", pt ) ;
    EXEC SQL
        UPDATE MOVIMIENTOS
        SET
            EST_MOVIMIEN = :sEST_MOVIMIEN,
                 COD_ERROR    = :sCOD_ERROR
        WHERE ROWID = :sMOV_ROWID ;
    oraperror ( "Update_MOVIMIENTO" ) ;
    return sqlca.sqlcode ;
}
```

Por último también cabe destacar que hemos optado por la creación de una función encargada de mostrar los valores de los distintos campos de un registro en BBDD. Esta función es muy útil para verificar que los registros han sido modificados cuando se ha realizado alguna acción sobre la tabla (inserción, modificación, borrado, etc.). Una vez más mostramos el ejemplo con la tabla de movimientos:

```
static int MostrarMovimiento(   char *s, movimiento *pt )
{
    if ( Dflag ) {
       MSJCom ("MOVIMIENTO: %s", s ) ;
       MSJCom ("sNUM_TELEFONO      =[%s]",   pt->sNUM_TELEFONO);
       MSJCom ("sNUM_TELNUEVO      =[%s]",   pt->sNUM_TELNUEVO);
       MSJCom ("NUM_SECUENCIA      =[%s]",   pt->sNUM_SECUENCIA) ;
       MSJCom ("COD_OPERACION      =[%s]",   pt->sCOD_OPERACION ) ;
       MSJCom ("NUM_TARJETA   =[%s]",   pt->sNUM_TARJETA ) ;
       MSJCom ("EST_MOVIMIEN       =[%s]",   pt->sEST_MOVIMIEN ) ;
       MSJCom ("TIP_LINEA          =[%s]",   pt->sTIP_LINEA ) ;
       MSJCom ("FEC_MOVIMIEN       =[%s]",   pt->sFEC_MOVIMIEN ) ;
       MSJCom ("COD_SERVICIO       =[%s]",   pt->sCOD_SERVICIO ) ;
       MSJCom ("COD_ERROR                =[%s]",   pt->sCOD_ERROR ) ;
```

```
        MSJCom ("CAUSA_BAJA                =[%s]",   pt->sCAUSA_BAJA ) ;
        MSJCom ("MULTISIM           =[%s]",   pt->sMULTISIM) ;
        MSJCom (“TIP_ACCION_SERVICIO=[%s]",   pt-TIP_ACCION_SERVICIO );
    }
    return 0 ;
}
```

6.4. ARRANQUE PROCESOS

Para proceder al arranque de los procesos se ha de diseñar un script que procederá a arrancar el binario compilado del proceso y generar el archivo de logs. Este binario se arrancará con un nohup para que se mantenga levantado continuamente.

El script de cada uno de los procesos a arrancar tiene la siguiente forma:

```
if [ $# != 1 ]; then
        echo "Forma de uso: $0 <NUM_INSTANCIAS>"
        exit 1
fi

PROC="NOMBRE DEL BINARIO GENERADO"

PATHLOG=`eval echo \\RUTA DEL ARCHIVO DE LOG`
PATHEXE=`eval echo \\RUTA EN LA QUE SE ENCUENTRA EL BINARIO GENERADO`

EXE=${PATHEXE}/${PROC}
LOG=${PATHLOG}/${PROC}.log


nohup ${EXE} ${1} 2> ${LOG} >&2 &

echo "Proceso = [${PROC}], Pid = [$!] "
exit 0
```

Este script recibe como parámetro el número de instancias que se deseen levantar de un proceso y se encarga de levantar el binario del proceso y generar el archivo de logs en la ruta que se indique en el script.

Para la parada de los procesos ejecutaremos la instrucción kill -11 PID que se encargará de parar el proceso.

6.5. ARRANQUE DE TUXEDOS

Para el arranque de los servidores tuxedos de la aplicación nos basamos en las instrucciones que están implementadas para la administración de este tipo de servidores.

Tmboot –s {NOMBRE DEL SERVIDOR}

Esta instrucción se encarga de levantar una instancia del servidor que se indique.

Si lo que se desea es detener el servidor se ejecutará la orden:

Tmshutdown –s {NOMBRE DEL SERVIDOR}.

Esta instrucción se encargará de detener el servidor. También se puede indicar que a través de la consola de administración que incorpora tuxedo (tmadmin) se pueden verificar una serie de valores: esta levantado el servidor, cuantas peticiones ha recibido, mediante el uso de que Gateway se puede enlazar. Todos estos valores se obtendrán mediante la ejecución del comando psc –s {NOMBRE DEL SERVIDOR}.

6.6. VARIABLES DE ENTORNO DE LA APLICACIÓN

Para el uso correcto de la aplicación se deben crear una serie de variables de entorno que se encargarán identificar los recursos necesarios para el mantenimiento de la aplicación.

VARIABLES DE ENTORNO NECESARIAS ORACLE:

- ORACLE_BASE=/aplicaciones/oracle/xxx/app/oracle => ubicación del TNSNAMES (archivo con los distintos esquemas de base de datos y cadena de conexión a los mismo)
- ORA_DUMP=/aplicaciones/Oracle/...
- ORACLE_SID= IDENTIFICADOR DEL ESQUEMA DE BASE DE DATOS
- ORACLE_TERM=vt220
- ORA_INSTANCE=/aplicaciones/oracle/... => instancia de ORACLE instalada
- ORACLE_HOME=/aplicaciones/oracle/... => ruta donde se encuentra instalada la distribución oracle

VARIABLES DE ENTORNO NECESARIAS TUXEDO

- PROV_MW_TUX_TRC=/... => ruta de log propias de TUXEDO (ULOG)
- TUXCONFIG=/.../ tuxedo/cfg/xxx_tx_config.tux => binario generado tras la compilación del ubbconfig.

Así mismo cabe destacar que será necesaria la creación de una serie de variables de entorno tanto para la parte de desarrollo como la de ejecución:

VARIABLES DESARROLLO:

- MAKERULES=/…/tolos => ruta donde se ubicarán las reglas de compilación necesarias para compilar la aplicación
- MAKE=make => alias para realizar la compilación de la aplicación.
- LINEA_BASE => variable que debe indicar el PATH donde está desplegada la aplicación

VARIABLES EJECUCION:

- XXX_LOG => ruta de carpetas donde se encuentran los logs de la aplicación.
- XXX_BIN => ruta donde se encuentran los binarios compilados de la aplicación.
- XXX_EJE => ruta donde se encuentran los distintos scripts de arranque de los procesos.

LINEA_EJECUCION => ruta que contiene todas las carpetas necesarias para la ejecución de la aplicación.

7. BIBLIOGRAFIA

7.1. BIBLIOGRAFIA FISICA

- John Larmouth. ASN.1 Complete. ISBN: 0-12-233435-3.
- Juan M. Andrade. The TUXEDO System: Software for constructing and managing Distributed Business Applications. ISBN: 0-201-63493-7.
- David Kurtz. PeopleSoft for the ORACLE DBA. ISBN: 978-1-4302-3708-2.
- Diego R. Llanos Ferraris. Fundamentos de informática y programación en C. ISBN: 978-84-9732-792-3.
- Javier Tuya. Técnicas cuantitativas para la gestión en la Ingeniería de Software. ISBN: 978-84-9745-204-5.

7.2. BIBLIOGRAFÍA WEB

- Documentación BEA Tuxedo: http://docs.oracle.com/cd/E13203_01/tuxedo/tux91/overview/overview.htm
- Manual ASN.1: http://www.obj-sys.com/asn1tutorial/asn1only.html
- Página ORACLE sobre documentación Pro*C: http://docs.oracle.com/cd/E11882_01/appdev.112/e10825/toc.htm
- Blog uso COCOMO para estimación de software: http://ingenieraupoliana.blogspot.com.es/2010/10/cocomo.html
- Foros ORACLE sistemas Tuxedo: https://community.oracle.com/community/fusion_middleware/transaction_processing/tuxedo
- Ejemplos diseño y modelación de proyectos software: http://www.monografias.com/trabajos28/proyecto-uml/proyecto-uml.shtml